Carolina Isabel Lagartinho de Oliveira

Licenciada em Ciências da Engenharia
Eletrotécnica e de Computadores

Edição de interfaces gráficas de utilizador e geração automática de componentes IOPT-Flow

Dissertação para a obtenção do Grau de Mestre em
Engenharia Eletrotécnica e de Computadores

Orientador: Doutor Filipe de Carvalho Moutinho, Professor Auxiliar, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Coorientador: Doutor Rogério Alexandre Botelho Campos Rebelo, Investigador, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Júri:

Presidente:	Doutor Luís Filipe Lourenço Bernardo
Arguentes:	Doutora Anikó Katalin Horváth da Costa
Vogais:	Doutor Filipe de Carvalho Moutinho

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2018

Edição de interfaces gráficas de utilizador e geração automática de componentes IOPT-Flow

Copyright © Carolina Isabel Lagartinho de Oliveira, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa.

A Faculdade de Ciências e Tecnologia e a Universidade Nova de Lisboa têm o direito, perpétuo e sem limites geográficos, de arquivar e publicar esta dissertação através de exemplares impressos reproduzidos em papel ou de forma digital, ou por qualquer outro meio conhecido ou que venha a ser inventado, e de a divulgar através de repositórios científicos e de admitir a sua cópia e distribuição com objetivos educacionais ou de investigação, não comerciais, desde que seja dado crédito ao autor e editor.

À comunidade científica

Agradecimentos

Primeiramente, e porque possibilitaram a feitura desta dissertação, apresento os meus agradecimentos aos meus orientadores Professor Filipe de Carvalho Moutinho e Doutor Rogério Alexandre Botelho Campos Rebelo. Os dois apresentaram-se sempre disponíveis e atentos, proporcionando um total apoio, para que fosse possível prestar este contributo.

Posteriormente, porque esta tese representa também o término de cinco anos após a minha chegada à FCT, agradeço o apoio dos meus, sempre, amigos, Bia, Morgado, Alex, Bruno, Pedro, David, Fernandes, e do meu querido afilhado Gil. E, porque são muitos, agradeço de forma abrangente a todos os meus colegas que me acolheram e com os quais tive a oportunidade de conviver.

Para além disso, agradeço a todos o meus amigos e membros do núcleo e da comissão pedagógica do curso, o NEEC e a CP-MIEEC, respetivamente, dos quais fiz parte, tendo presenciado com muito apreço o seu crescimento e reconhecimento perante o curso e a comunidade académica.

Do mesmo modo, agradeço a todos os meus professores, que contribuíram para a minha integração e aprendizagem.

Por fim, agradeço a minha família, especialmente à minha avó e ao meu sobrinho.

Resumo

A presente dissertação propõe um editor de interfaces gráficas de utilizador e um gerador de componentes IOPT-Flow, nomeado IOPT-Flow GUI Editor. O ambiente IOPT-Flow GUI Editor foi desenvolvido como parte integrante do ambiente IOPT-Flow Editor, por sua vez, dedicado ao desenvolvimento de controladores de sistemas embutidos e ciberfísicos, onde os modelos são desenhados tendo como base o formalismo DS-Pnet.

Existem várias linguagens que permitem a descrição de interfaces gráficas de utilizador e, tipicamente, baseiam-se no formato de representação XML. Como os modelos e componentes DS-Pnet são armazenados em documentos XML e o ambiente IOPT-Flow Editor trabalha diretamente nesses documentos, a nova ferramenta permite o desenho de interfaces gráficas de utilizador, possibilitando também a sua conversão para modelos e componentes IOPT-Flow através da geração automática de documentos XML. Assim, as interfaces criadas podem posteriormente ser usadas no ambiente IOPT-Flow Editor, tendo ao seu dispor um conjunto de ferramentas de automatização de projeto oferecidas por esse ambiente, nomeadamente de geração automática de código.

Ao longo das validações efetuadas foi possível comprovar a correta implementação do ambiente desenvolvido, analisando-se o comportamento das interfaces criadas, das quais se destaca uma interface que pode ser aplicada à visualização e monitorização de parâmetros associados ao funcionamento de uma cadeira de rodas elétrica, transpondo o uso do novo ambiente para outro tipo de aplicações e industrias, incluindo o seu uso a nível académico.

Palavras- chave: DS-Pnet, Editor de GUI, Gerador automático de componentes, Gerador de código C automático, IOPT-Flow, monitorização, representação XML, simulação remota, sistemas embutidos e ciberfísicos.

Abstract

This dissertation proposes a graphical user interface editor and an IOPT-Flow component generator, named IOPT-Flow GUI Editor. The IOPT-Flow GUI Editor was developed as integrant part of IOPT-Flow Editor, an environment dedicated to the development of embedded and distributed cyber-physical systems, which models are designed based on DS-Pnet modeling formalism.

There are several languages that allow the description of graphical user interfaces and they are typically based on XML representation format. Because DS-Pnet models and components are stored in XML documents and the IOPT-Flow Editor works directly on them, the new tool allows the design of graphical user interfaces, and at the same time enables them to be converted to IOPT-Flow models and components, through the automatic XML documents generation. In this way, the created interfaces can later be used on IOPT-Flow Editor environment, that provides a set of design automation tools, like automatic code generation.

With validations carried out it was possible to verify the correct implementation of the developed environment by analyzing the behavior of interfaces created, like an interface that can be applied to the display and monitoring of parameters of an electric wheelchair, transposing the use of the new environment to another kind of applications and industries, including its use at the academic level.

Keywords: DS-Pnet, GUI Editor, Automatic Component Generator, Automatic C code generator, IOPT-Flow, monitoring, XML representation, remote simulation, embedded and cyber-physical systems.

Conteúdo

Lista de Figuras

Lista de Tabelas

Siglas e Acrónimos

ANSI	American National Standards Institute
ARM	Advanced RISC Machine
AST	Abstract Syntax Tree
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
CUDABlock	Compute Unified Device Architecture Block
DOM	Document Object Model
DS-Pnet	Dataflow, Signals and Petri nets
GEF	Graphical Editing Framework
GPIO	General Purpose Input/Output
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IO	Input-Output
IOPT	Input-Output Place-Transition
IOPT-Tools	Input-Output Place-Transition-Tools
IOPT-Flow	Input-Output Place-Transition-Flow
JSON	JavaScript Object Notation
LabVIEW	Laboratory Virtual Instrumentation Engineering Workbench
MIT	Massachusetts Institute of Technology

PHP	Hypertext Preprocessor
PNML	Petri-net markup language
RAD	Rapid Application Development
SCADA	Supervisory Control and Data Acquisition
SD	Secure Digital Card
SFTP	SSH File Transfer Protocol
SGML	Standard Generalized Markup Language
SVG	Scalable Vector Graphics
TNG	The Next Generation
UI	User Interface
VHDL	VHSIC Hardware Description Language
WinSCP	Windows Secure CoPy
XML	eXtensible Markup Language
XSLT	eXtensible Stylesheet Language for Transformation

1

Introdução

Neste capítulo é introduzido o tema estudado no âmbito da dissertação para a obtenção do grau de mestre em engenharia eletrotécnica e de computadores. Primeiramente, são apresentadas as motivações; posteriormente, são listados os objetivos seguidos das contribuições provenientes do trabalho desenvolvido; e por fim, é apresentada a estrutura deste documento que permitirá elucidar o leitor sobre o conteúdo da dissertação.

1.1.Motivações

Nesta secção são apresentados dois tópicos nos quais assentam as motivações para o tema: uma motivação científica para a contribuição na edição e geração automática de interfaces gráficas de utilizadores, GUI (*Graphical User Interface*); e uma motivação social, que promoverá a integração de uma GUI num sistema de monitorização para cadeiras de rodas elétricas.

Este trabalho apoia o estudo e uso de ferramentas de automatização de projeto no desenvolvimento de GUI para sistemas embutidos e ciberfísicos. As ferramentas foram disponibilizadas pelo ambiente de modelação IOPT-Flow (*Input-Output Place-Transition*)-Flow Editor, dedicado ao desenvolvimento de controladores de sistemas embutidos e ciberfísicos, onde os modelos são desenhados tendo como base o formalismo de modelação híbrido DS-Pnet (*Dataflow, Signals and Petri nets*), que combina redes de Petri e elementos *dataflow* [1], [2]. Num sistema modelado em DS-Pnet, a implementação da máquina de estados é realizada com redes de Petri, e as operações de processamento e manipulação de sinais e eventos de entrada são feitas através de componentes *dataflows*, que produzem sinais e eventos de saída.

Existem ferramentas que usam editores gráficos de redes de Petri, permitindo que através dos controladores que são desenhados seja apresentada a interface gráfica que lhes corresponde: o comportamento do controlador é descrito através de modelos de redes de Petri, e as ferramentas

suportam as características da interface gráfica, associando-as às características do modelo comportamental; duas das principais ferramentas são o *Animator* e o *Synoptic* [3], [4]. O *Animator* permite uma definição interativa das características gráficas do sinóptico; o *Synoptic* é responsável pelo controlo-execução da rede de Petri e pela atualização da GUI em tempo real.

Ainda a título de motivação, considere-se a pesquisa e o trabalho desenvolvido no âmbito do estudo de cadeiras de rodas elétricas; tipicamente assenta:

(1) No melhoramento do sistema de controlo, nomeadamente a sua personalização considerando a patologia do utilizador e a sua capacidade de interação com o sistema;
(2) Na avaliação da melhor trajetória ou a análise do ambiente envolvente e dos obstáculos detetados, aumentando a capacidade de mobilidade do utilizador;
(3) No projeto de soluções que otimizam o funcionamento dos motores, da bateria e o custo monetário da cadeira;
(4) Na necessidade de monitorização dos sinais vitais do utilizador para efeitos de tratamento médico, entre outros.

Analogamente, é possível idealizar-se um sistema monitor dos parâmetros associados ao funcionamento da cadeira, apresentados através de uma GUI; os objetivos serão idênticos, permitindo também facilitar a deteção de erros e falhas no sistema, acelerando a respetiva resolução por parte dos técnicos.

Assim, considerando o que foi exposto imediatamente acima, esta dissertação propõe o desenvolvimento de um ambiente dedicado à edição e geração automática de interfaces gráficas de utilizador. A nova ferramenta, nomeada IOPT-Flow GUI Editor, permite realizar o desenho de interfaces gráficas de utilizador, e definir a sua relação com controladores editados no ambiente IOPT-Flow Editor através da geração automática de componentes IOPT-Flow; ou seja, é um editor de GUI que gera componentes IOPT-Flow, associando-os ao modelo de um controlador. Para além disso, também faz o aproveitamento das ferramentas de automatização de projeto disponíveis no ambiente IOPT-Flow Editor, como por exemplo, os geradores de código C e JavaScript, que podem ser usados para controlar os objetos gráficos da GUI.

1.2. Objetivos

Tendo como base os motivos em que se sustenta esta dissertação, atente-se aos objetivos atinentes que se apresentam de seguida.

Primeiramente, pretende-se estabelecer e automatizar tanto quanto possível as relações entre as características de modelos comportamentais de sistemas embutidos e os atributos das interfaces gráficas respeitantes, permitindo a edição e geração automática de GUI através do ambiente IOPT-Flow GUI Editor; ou seja, utilizar o novo ambiente para a edição de GUI e posterior geração do seu modelo, e, por conseguinte, componente IOPT-Flow, que pode ser associado a outros modelos e componentes desenvolvidos no ambiente IOPT-Flow Editor.

Posteriormente, é utilizada a ferramenta desenvolvida para suportar a implementação de um sistema de monitorização para cadeiras de rodas elétricas.

1.3. Contribuições

Conhecidas as motivações e apresentados os objetivos, considerem-se as contribuições da dissertação: (1) elaboração de um levantamento bibliográfico sobre ferramentas que permitem a prototipagem de interfaces gráficas; (2) desenvolvimento de uma ferramenta que permita a edição de GUI e geração de componentes IOPT-Flow para serem usados no ambiente IOPT-Flow Editor; (3) estudo do desempenho da ferramenta aplicada a um sistema de monitorização de cadeiras de rodas elétricas, permitindo validar o comportamento do sistema implementado, tendo em consideração os sistemas referidos na literatura.

1.4. Estrutura do documento

Este documento esta organizado e estruturado em cinco capítulos, incluindo o presente; são eles: *Introdução*, *Estado da Arte*, *Proposta e Ambiente Desenvolvido*, *Testes e Exemplos de Validação*, *e Considerações Finais*.

O segundo capítulo, *Estado da Arte*, apresenta alguns conceitos e definições inseridas no tema da dissertação e que permitem avaliar melhor os conteúdos relacionados com a mesma, e analisar um conjunto de aplicações que também são apresentadas no capítulo.

No terceiro capítulo, denominado *Proposta e Ambiente Desenvolvido*, é realizada uma descrição pormenorizada sobre o desenvolvimento do ambiente IOPT-Flow GUI Editor, nomeadamente a sua interação com o ambiente IOPT-Flow Editor, a caracterização e composição do ambiente, as ferramentas que oferece, e modo de interação do utilizador com o mesmo.

O capítulo *Testes e Exemplos de Validação* é o quarto capítulo, e apresenta a fase de testes e validação da implementação do editor e gerador automático de interfaces gráficas de utilizador. Para validar o novo ambiente, foram preparados dois exemplos de aplicação, dos quais se destaca a criação de uma interface que pode ser aplicada à visualização e monitorização de parâmetros associados ao funcionamento de uma cadeira de rodas elétrica.

O quinto e último capítulo deste documento, *Considerações Finais*, apresenta uma breve síntese do trabalho desenvolvido, realizando-se uma reflexão sobre os resultados obtidos durante a implementação do IOPT-Flow GUI Editor, bem como aquando da sua validação, durante a fase de testes e validações do ambiente.

Por fim, no final do documento são elencadas as referências que foram consultadas, e são apresentados anexos com conteúdo respeitante a uma pequena parte do código desenvolvido, e alguns de resultados da geração automática de componentes IOPT-Flow.

2

Estado da Arte

Intitulado de *Estado da Arte*, o presente capítulo foca o seu conteúdo na análise de alguns ambientes, aplicações e programas de desenvolvimento de interfaces gráficas de utilizador, GUI. Inicialmente, e de forma relevante, é apresentado o conceito de linguagem visual; de seguida, é realizada uma da análise sobre a representação de GUI; depois, são apresentados alguns ambientes de desenvolvimento de GUI; posteriormente, relaciona-se o trabalho da dissertação com o conceito de sistemas de supervisão, controlo e aquisição de dados, SCADA (*Supervisory Control and Data Acquisition*); seguidamente, são apresentados o formalismo de modelação DS-Pnet e o ambiente IOPT-Flow para o desenvolvimento de controladores de sistemas embutidos e reconfiguráveis; por fim, é realizada uma breve análise sobre o mercado mundial de cadeiras de rodas elétricas.

2.1. Linguagem visual

Principie-se o *Estado da Arte* analisando o que é a linguagem visual no âmbito da programação. A comunidade de pesquisa sobre a linguagem visual não possui uma definição única e universalmente aceite sobre o que é exatamente a linguagem visual [5]. Para trabalhos relacionados com informação e visualização científica o objetivo geral é, vagamente, codificar dados numa representação visual, podendo-se argumentar que a visualização é uma linguagem visual para se poder comunicar sobre dados; no entanto, serão as técnicas de avaliação da visualização apropriadas para avaliar linguagens visuais de todo o tipo?

Em *Journal of Visual Languages and Computing*, [5], M. Erwig, K. Smeltzer e X. Wang propõem uma ontologia que, sendo caracterizada por um conjunto de *tags*, permite descrever aspetos de uma linguagem visual quanto à aparência sintática e à semântica de notação. Estes *tags* podem ser usados para fornecer mais detalhes e distinguir diferentes notações [5].

Para se obterem descrições mais sucintas de determinadas classes de linguagens visuais é útil considerar-se a expansão da ontologia por meio da derivação de *tags*, permitindo a identificação uma categoria particular de linguagens visuais. A adição de *tags* corresponde assim a uma operação de interseção.

2.2.Representação interfaces gráficas de utilizador

Existem várias linguagens que permitem a descrição de interfaces gráficas de utilizador. Tipicamente, estas linguagens baseiam-se no formato XML (*eXtensible Markup Language*), e os documentos que definem as GUI podem ser interpretados e visualizados por diferentes programas, como aplicações *web* [6].

XML é uma notação usada para a representação de estruturas de dados que, tal como o HTML (*Hypertext Markup Language*), deriva do SGML (*Standard Generalized Markup Language*). Tal como o nome indica, o XML é extensível através do uso de *tags*, sendo benéfico para o desenvolvimento de diversas aplicações [7] e troca de dados: dois programas que sejam executados em plataformas diferentes podem partilhar dados em XML se respeitarem as *tags* definidas. Assim, uma GUI tanto pode ser descrita sob a forma de instruções de programa, ou através de documentos que permitem personalizar e definir interfaces.

Para minimizar a necessidade de programação no processo de criação de GUI, existem muitas ferramentas usadas para a sua edição; no geral, estas ferramentas permitem que se desenhe e se gere o código para GUI, que pode estar integrada na aplicação que está a ser desenvolvida, como acontece no Visual Studio da Microsoft, por exemplo. Do mesmo modo, existem algumas tecnologias cujo relacionamento entre os documentos e as interfaces gráficas se baseiam na estratégia de enriquecimento dos documentos, fornecendo-lhes recursos básicos para a interação com utilizador. Por exemplo, os controlos da GUI estão ligados à lógica do programa por eventos, que geralmente são ações executadas por um utilizador sobre o GUI, e o documento contém informações sobre qual a parte do programa que deve ser executado para cada evento.

2.3.Compute Unified Device Architecture Block

CUDABlock (*Compute Unified Device Architecture Block*) é uma ferramenta que foi desenvolvida com base no OpenBlocks, e que permite a programação gráfica para CUDA (*Compute Unified Device Architecture*); através de uma interface gráfica de utilizador, facilita a programação paralela em sistemas de computador com vários núcleos (*multicore*).

OpenBlocks, desenvolvido por um grupo de investigação do MIT (*Massachusetts Institute of Technology*), é um biblioteca extensível (*open source*) de código Java e uma *framework* inspirada no StarLogo TNG (*The Next Generation*) que é usada para programação de interfaces de utilizador, UI (*User Interface*), tal que, apoia as tarefas de programação por blocos (*blocks*) pré-definidos [8],

[9]. Para além disso, através da especificação de apenas um ficheiro XML, permite que cada programador construa e itere o seu próprio sistema de programação por blocos; assim, os programadores podem focar-se mais no desenho dos sistemas ao invés nos detalhes de implementação [8].

OpenBlocks é composto por dois pacotes (*packages*): o *codeblocks*, que é responsável por garantir a maioria das funcionalidades da biblioteca, e o *slcodeblocks*, respeitante ao código desenvolvido num projeto denominado StarLogo TNG [9].

A Figura 2.1 apresenta o ambiente de programação OpenBlocks; o ambiente de programação ou a área de trabalho é dividido em múltiplos componentes de interface de utilizador chamados *widgets* [8].

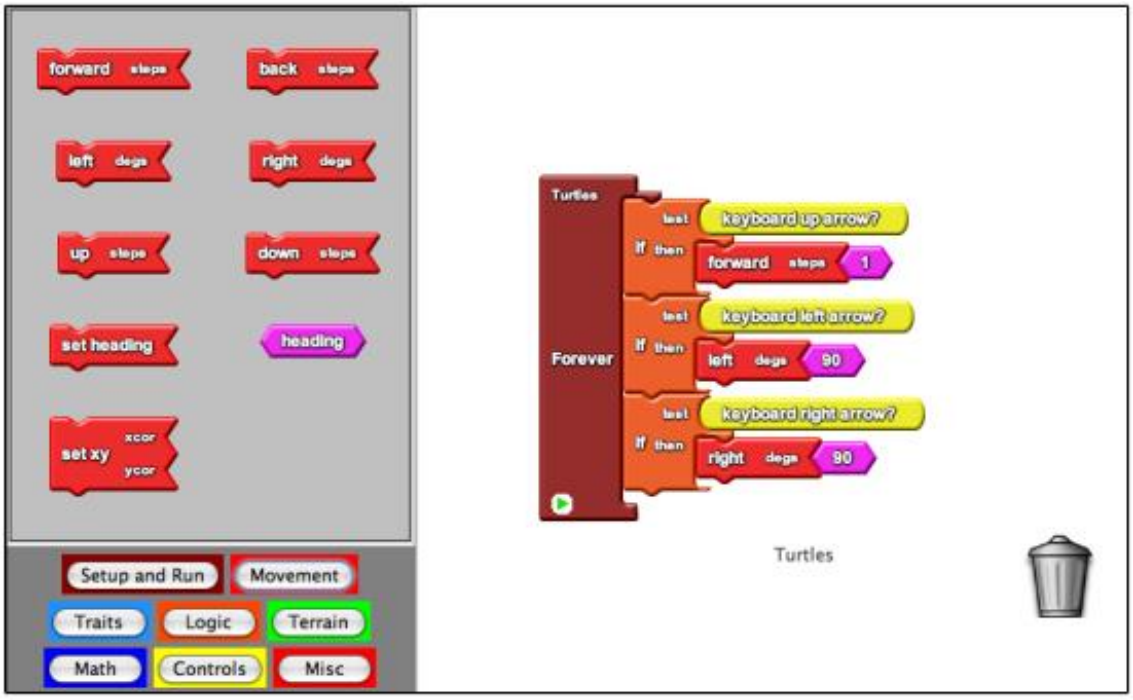


Figura 2.1: Ambiente de programação OpenBlocks.

A área superior esquerda permite que o utilizador possa escolher diferentes blocos de programação, a região abaixo exibe as diferentes categorias de blocos disponíveis, e à direita, na área de exibição é onde se realiza e visualiza a programação, arrastando-se os blocos do lado esquerdo, conectando-os à direita; os programadores apenas necessitam de arrastar os blocos para especificar ações e, em seguida, conectar os blocos selecionados de acordo com o algoritmo que se deseja implementar [10].

CUDABlock adicionou ao OpenBlocks dois novos conjuntos de blocos para a programação em CUDA: Blocos ANSI (*American National Standards Institute*) C e Blocos CUDA; e dois blocos de otimização para oferecer suporte a duas técnicas de otimização básicas e muito utilizadas em CUDA: *share block*, para partilha de memória *cache*, e *blocking block*, para *tiling*.

Os Blocos ANSI C são usados para a programação em C, nomeadamente os blocos *control*, *test*, *math*, *system* IO (*Input-Output*), *variables/constants* e *code*; os blocos *code* possibilitam a construção personalizada de código C, permitindo a definição de blocos de código especializado.

Do mesmo modo, os blocos CUDA suportam operações básicas de álgebra linear com vetores e matrizes: adição, subtração, multiplicação e operações de convolução. A principal diferença entre os blocos ANSI C e os blocos CUDA é que os primeiros serão mapeados ou convertidos para código C que será executado no CPU (*Central Processing Unit*), e os últimos estão relacionados com os núcleos do CUDA, que serão executados no GPU (*Graphics Processing Unit*) [10].

Assim, o CUDABlock é uma abordagem mais intuitiva que pode reduzir significativamente a complexidade da programação paralela em CUDA [10], pelo que pode ser usado como uma ferramenta de rápida prototipagem de aplicações GPU ou como uma plataforma para aprendizagem de programação paralela, visto que a ferramenta é capaz de exibir os programas gerados no CUDA.

2.4. Laboratory Virtual Instrumentation Engineering Workbench

Com o rápido desenvolvimento de sensores e vários componentes de automação, o processo de teste pode ser automatizado usando programas como o LabVIEW [11] (*Laboratory Virtual Instrumentation Engineering Workbench*), uma linguagem de programação gráfica utilizada para a aquisição e visualização de dados, seja em setores da indústria ou projetos de pesquisa em laboratórios académicos.

Efetivamente, o LabVIEW é um ambiente altamente produtivo, usado para o desenvolvimento gráfico de algoritmos, tal que, fornece uma variedade de recursos e ferramentas, destacando-se devido à sua linguagem de programação gráfica, G, ao seu compilador integrado, ao seu *linker* e às suas ferramentas de depuração. Estas ferramentas facilitam as ações repetitivas, onde os principais benefícios são economizar tempo e esforço, bem como a precisão e exatidão no desenvolvimento de códigos complexos [12]. A linguagem de programação G é um modelo de programação bastante intuitivo na forma de fluxo de dados, semelhante a um fluxograma; possibilita a programação orientada a objetos; e apresenta uma curva de aprendizagem mais curta, comparando com a programação textual.

Comparando o LabVIEW com programações de texto, o primeiro apresenta-se mais amigável por ser possível desenvolver-se um programa conectando-se vários blocos através de ligações [11]; possui um painel frontal para a preparação da GUI, e um painel com um diagrama de blocos para programação gráfica da GUI [11] – Figura 2.2. No painel frontal podem ser distribuídos controlos e indicadores genéricos, como cadeias de caracteres (*strings*), numéricos e botões; ou controlos e indicadores técnicos, como gráficos, tabelas, termómetros e escalas.

O objetivo do diagrama de blocos é separar o código-fonte da interface de utilizador de forma lógica e simples: os objetos do painel frontal aparecem como terminais no diagrama de blocos; os terminais no diagrama de blocos refletem as alterações feitas aos respetivos objetos no painel frontal, e vice-versa [13]. Assim, os controlos e indicadores colocados no painel frontal são

convertidos para blocos no diagrama de blocos, onde se torna possível modelar o comportamento do sistema tendo como base metodologias de modelação de processos e tarefas como diagrama de blocos, ou fluxogramas que respeitam regras de fluxo de dados (*dataflow*).

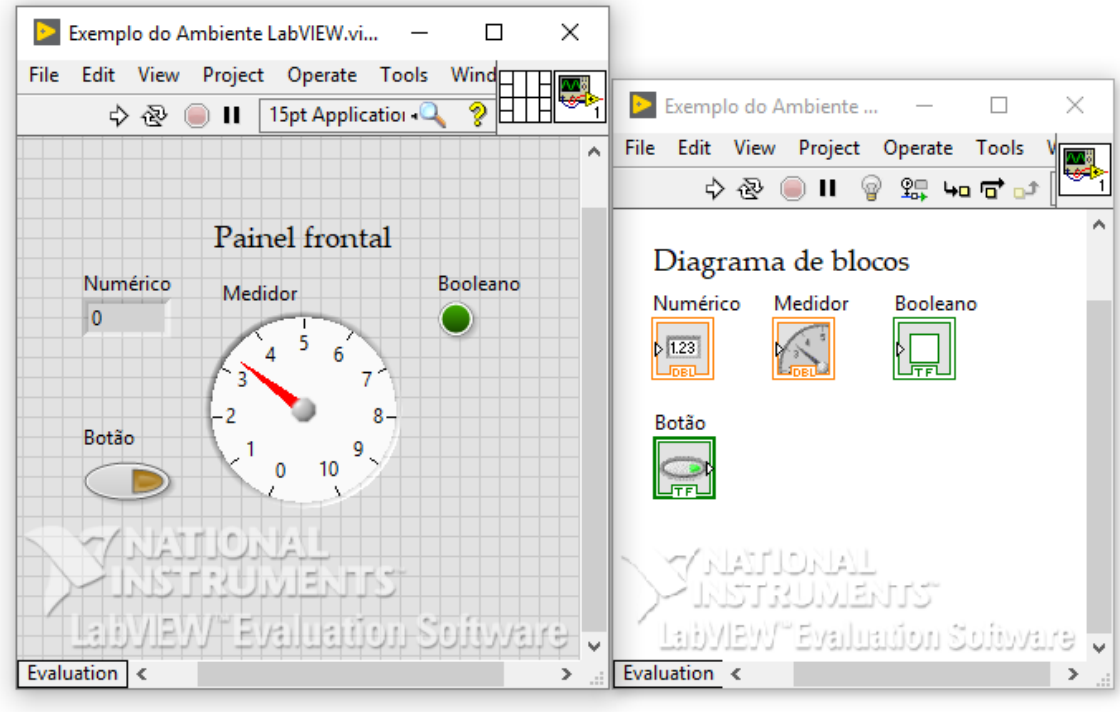


Figura 2.2: Ambiente LabVIEW, onde à esquerda é apresentado o painel frontal, e à direita o diagrama de blocos.

2.5. Ambientes de desenvolvimento integrados

Ambientes de desenvolvimento integrado, IDE (*Integrated Development Environment*), como Microsoft Visual Studio, Eclipse, NetBeans, WebStorm, PyCharm, Android Studio, entre outros, dedicam-se ao desenvolvimento rápido de aplicações, RAD (*Rapid Application Development*), possuindo componentes e ferramentas dotadas para o desenho de interfaces gráficas de utilizador [14].

O Microsoft Visual Studio foi criado pela Microsoft e é usado para desenvolver *web sites*, aplicações *web* e serviços *web*. Permite que o editor de código e o depurador suportem (a vários graus) diversas linguagens de programação [15], incluindo C, C++ e C++/CLI (através do Visual C++), VB.NET (através do Visual Basic .NET), C# (através do Visual C#) e F# (a partir de Visual Studio 2010).

Das várias ferramentas que o Visual Studio possui, destaque-se o *design* de *forms* dedicados à elaboração de aplicações do tipo interface gráfica de utilizador, GUI, *design web*, *design* de classes e de esquemas de base de dados [15]. No Visual Studio, um *form* é a representação de qualquer janela criada a partir das propriedades, métodos e eventos disponibilizados pela classe *Form Class*.

As propriedades disponíveis permitem alterar o tamanho, a posição, a cor e a aparência da caixa de diálogo ou da janela, bem como dos controlos colocados no *form*; os métodos da classe apoiam a manipulação do *form*, tal que, o método *ShowDialog* apresenta o *form* como uma caixa de diálogo e o método *SetDesktopLocation* posiciona-o no ambiente de trabalho; por fim, os eventos permitem que se dê resposta a ações efetuadas sobre o *form*, como por exemplo, atualizar a informação disponibilizada [16].

A Figura 2.3 apresenta um exemplo simples de como a um *design* de uma GUI corresponde um código que, sendo gerado automaticamente, possui todas as inicializações e fornece todas as configurações respeitantes a cada um dos controlos colocados na interface.

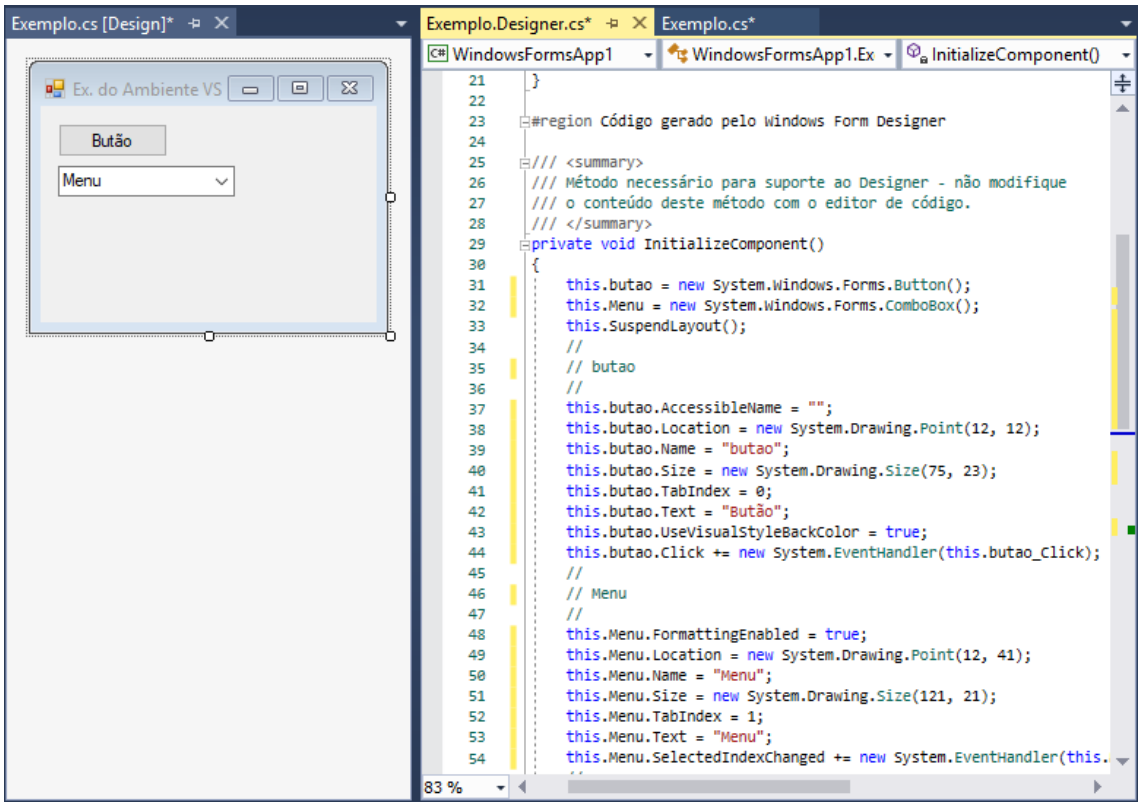


Figura 2.3: Ambiente Visual Studio, onde à esquerda é apresentado o desenho de uma GUI, e à direita o respetivo código, gerado automaticamente.

Seguidamente apresenta-se o WindowBuilder. O WindowBuilder é uma ferramenta oferecida pelo Eclipse e facilita o desenvolvimento em Java de aplicações do tipo GUI através de *forms*, sendo o código Java gerado automaticamente.

Tal como no exemplo anterior, esta ferramenta baseia-se no conceito *drag-and-drop*, sendo também caracterizada por um conjunto de controlos que podem ser adicionados ao *form* e associados a eventos [17], [18].

O WindowBuilder é extensível e personalizável, ou seja, permite que através da adição de novas extensões e *toolkits* seja possível desenvolver-se novos *designs* de GUI. A comunicação entre a GUI e os documentos que a caracterizam é apoiada pela representação em árvore, AST (*Abstract*

Syntax Tree), usada para navegar pelo código-fonte; por outro lado, o GEF (*Graphical Editing Framework*) é usado para exibir e fazer a gestão da apresentação visual [17], [18].

Idêntica à Figura 2.3, a Figura 2.4 apresenta um exemplo do desenho de uma GUI e do código que, sendo gerado automaticamente, inicializa e configura as características respeitantes a cada um dos controlos colocados na GUI.

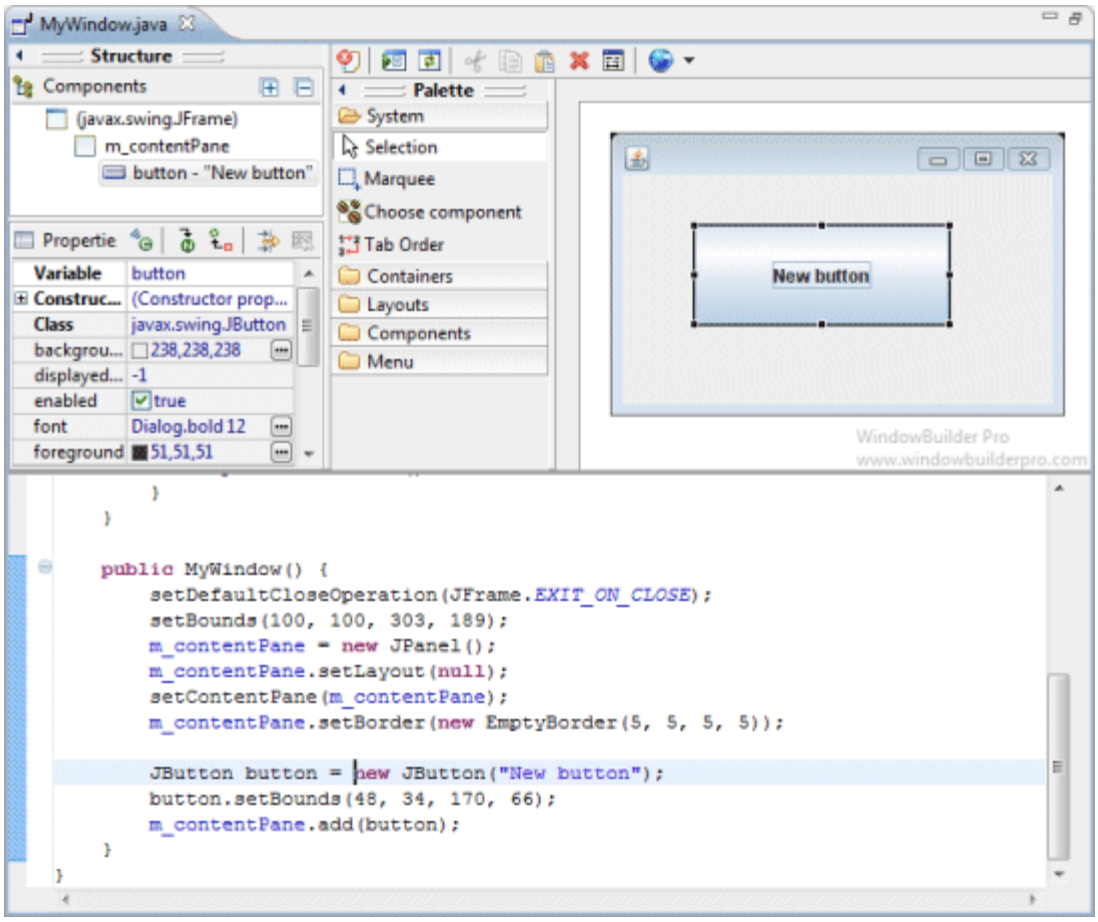


Figura 2.4: Ambiente WindowBuilder, onde em cima é apresentado o desenho de uma GUI, e em baixo o código associado, gerado automaticamente.

2.6.Ferramentas Animator e Synoptic

O *Animator* e o *Synoptic* [3], [19] são duas ferramentas que em conjunto permitem a geração automática de interfaces gráficas de utilizador para sistemas embutidos, definindo a relação que é estabelecida entre as características do modelo comportamental de um sistema, e o estado dos atributos e elementos gráficos da respetiva interface gráfica de utilizador; e isso é garantido sem ser necessário qualquer escrita de código.

Efetivamente, enquanto o *Animator* define as características da interface gráfica que são apresentadas no *layout* gráfico, e é responsável pela criação de um conjunto de regras que permitem associar essas características às características de modelos de rede de Petri IOPT [20], [21], ; o

Synoptic, por sua vez, é responsável pelo controlo e execução do modelo, e atualização da interface gráfica criada [3], [4], [22]. Assim o principal objetivo do *Animator* é a preparação de um conjunto de ficheiros XML para que posteriormente possam ser usados pelo *Synoptic*; o *Synopitc* dependerá dos ficheiros que são preparados pelo *Animator*, que incluem a especificação das características do ambiente, e a caracterização das regras que associam a rede de Petri do modelo às características da interface gráfica de utilizador, entre outros. Atente-se à Figura 2.5.

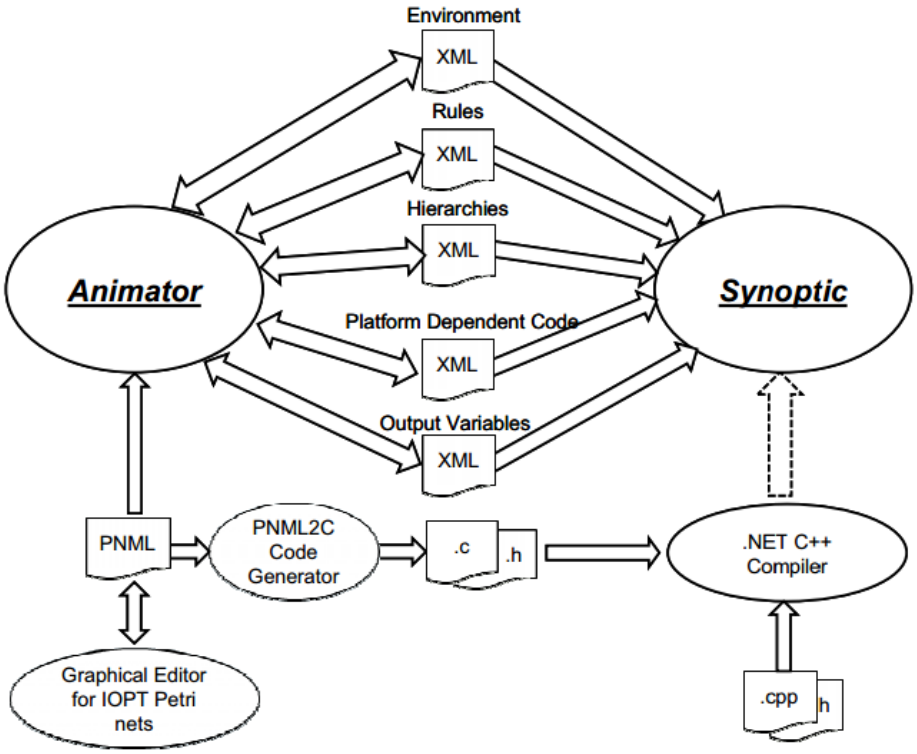


Figura 2.5: Esquema da interação entre o *Animator* e o *Synoptic*.

Como se pode observar na Figura 2.5, inicialmente – com início no canto inferior esquerdo – é necessária a criação do modelo comportamental de um sistema, através de um editor gráfico de modelos de redes de Petri IOPT; e, de seguida, é gerada a sua representação em PNML (formato de representação baseado em XML, para redes de Petri), que contém todas as características da rede de Petri IOPT [21].

Depois de criado o modelo, são criados os vários *layouts* e definido o aspeto das janelas da interface através do uso de um conjunto de ícones, imagens ou outro tipo de ficheiros multimédia; as imagens podem ser aumentadas e dispostas em diferentes sítios e pode-se definir uma hierarquia entre elas. Estas caracterizações – do ambiente, e da hierarquia – são representadas por ficheiros XML. A Figura 2.6 apresenta o ambiente e *layout* gráfico criado para um exemplo, onde foi modelado o funcionamento de um parque de estacionamento.

De seguida, são definidas as regras entre o *layout* gráfico e o modelo: para cada elemento pertencente ao *layout* é associada a uma parte da rede de Petri do modelo em causa, e as regras vão descrever o comportamento desses elementos consoante o estado da rede. Para que seja pos-

sível definirem-se regras que respeitam determinadas condições, as informações que estão contidas no ficheiro PNML – que representa o modelo IOPT – são carregadas no *Animator*. Tanto as características estáticas como as características dinâmicas do modelo são consideradas pelas regras: as primeiras estão relacionadas com as marcas nos lugares da rede de Petri e com estado dos sinais de entrada e de saída no modelo; as dinâmicas dizem respeito ao disparo de transições da rede e à ocorrência de eventos [20], [21], [23]. Considerando isto, é possível criar-se um conjunto de regras especificas para uma determinada aplicação. As regras são apresentadas num ficheiro XML.

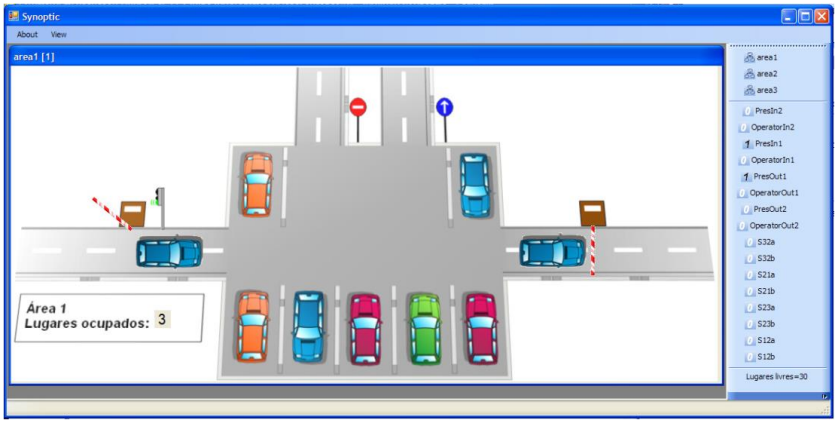


Figura 2.6: Apresentação do *Synoptoc*, onde foi modelado o funcionamento de um parque de estacionamento.

Efetivamente, as regras vão descrever o comportamento da GUI, quando é detetada a ocorrência de uma determinada condição no modelo IOPT; ou seja, a interface gráfica de utilizador vai ser automaticamente atualizada aquando da ocorrência de um evento produzido pela mudança do estado da rede de Petri do modelo. Cada regra é definida segundo o formato "se antecedente, então consequente", onde o antecedente depende da característica da rede de Petri do modelo, e o consequente identifica os efeitos que ocorrem na interface gráfica [4] – Figura 2.7.

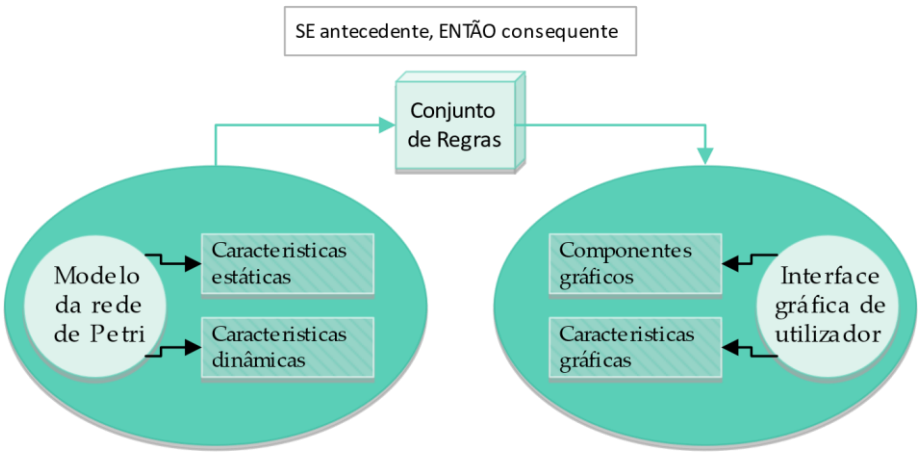


Figura 2.7: Esquema ilustrativo da avaliação e aplicação das regras estabelecidas entre um modelo e a respetiva interface.

Assim, durante a execução de um sistema, as respetivas regras são constantemente avaliadas e permitem alterar o estado dos componentes da interface, apresentados no *layout* gráfico suportado pelo *Synoptic*. Através do gerador de código C automático, é possível traduzir o ficheiro PNML, nomeadamente a rede de Petri que modela o sistema, e usar esse código para executar o modelo [24], [25]. Os arquivos C gerados, ".c" e ".h" são vinculados aos arquivos de projeto *Synoptic* (previamente preparados), para construir o arquivo executável do aplicativo *Synoptic*. O *Synoptic* é capaz de ler os dados da configuração descrita e armazenada nos ficheiros XML, e executar o modelo de rede de Petri, verificando mudanças, determinando o próximo estado; e atualizar automaticamente a GUI de acordo com as regras definidas [3], [4], [22], [26], [27].

2.7. Sistemas de supervisão, controlo e aquisição de dados

Atente-se o conceito SCADA. SCADA são sistemas que permitem a supervisão, o controlo e a aquisição de dados de um determinado sistema [27] e têm sido usados em várias indústrias que requerem a automação de processos. Os sistemas SCADA facilitam operações de monitorização; possuem interfaces que permitem realizar monitorização e fornecer comandos, tais como alterações de dados de *set-point*, e exibem resultados da medição efetuada por instrumentos usados num sistema; o processo é realizado em tempo real e é executado por sensores e atuadores que comunicam com o sistema através da rede [28].

No âmbito dos trabalhos desenvolvidos nesta dissertação, considerar-se-á a prototipagem de uma solução do tipo SCADA, um sistema que pode ser implementado em múltiplas plataformas microcontroladas e interagir com o processo – a cadeira de rodas elétrica – que é monitorizado e controlado via redes de comunicação de dados.

A interface do sistema de monitorização de cadeiras de rodas elétricas será desenvolvida tendo como base metodologias dedicadas ao desenvolvimento de sistemas embutidos, que com o suporte de um conjunto de ferramentas de automatização de projeto, podem integrar a geração automática da GUI associada.

Efetivamente, pretende-se estabelecer e automatizar tanto quanto possível as relações entre as características de modelos comportamentais de sistemas embutidos e os atributos das interfaces gráficas respeitantes, permitindo a edição e geração de GUI através de componentes gerados com um editor, considerando um modelo DS-Pnet; ou seja, um editor de GUI que gera componentes associados a um modelo e que podem ser usados para gerar o código C e JavaScript da respetiva do modelo a interagir com a GUI.

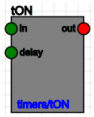
2.8. Formalismo de modelação DS-Pnet e ambiente de desenvolvimento IOPT-Flow

Nesta secção são apresentados o formalismo de modelação DS-Pnet e o ambiente de desenvolvimento IOPT-Flow.

O formalismo de modelação DS-Pnet foi criado para suportar o desenvolvimento de sistemas embutidos e ciberfísicos através da combinação de redes de Petri e elementos *dataflow* [29]. As partes de controlo são modeladas com redes de Petri, usadas para a implementação de máquinas de estados; e as operações de manipulação de dados que processam sinais e eventos de entrada são expressas através de *dataflows* [2], [30]. Os modelos desenhados com este formalismo podem ser compostos pelos itens apresentados na tabela que se segue – Tabela 2.1.

Os lugares, as transições e os arcos da rede de Petri comportam-se como nós de uma tradicional rede de Petri de baixo nível, e tipicamente são usados para modelarem as partes reativas dos controladores. A interface externa de um modelo é definida por sinais e eventos de entrada e saída, usados para ler sensores, definir o valor de atuadores de saída, ou comunicar com outros subsistemas. A parte do fluxo de dados de um modelo é definida por uma rede de operações (*node*) conectadas por arcos *dataflow* (*read-arc*), usados para ler valores de sinais e eventos de entrada, resultados de outras operações e o estado da rede de Petri [31].

Tabela 2.1: Componentes do formalismo de modelação DS-Pnet.

	Lugar da rede de Petri (*place*)
	Transição da rede de Petri (*transition*)
	Arco da rede de Petri (*arc*)
	Arco dataflow (*read-arc*)
	Sinal de entrada (*input signal*)
	Evento de entrada (*input event*)
	Sinal de saída (*output signal*)
	Evento de saída (*output event*)
	Operação dataflow (*node*)
	Componente

É interessante perceber que as transições da rede de Petri podem ser inibidas por condições e eventos impostos pelo *dataflow*, e que as operações do *dataflow* podem depender de valores baseados no estado da rede de Petri [1] (marcas em lugares e disparos de transições). Esta combinação simplifica o processo de modelação.

A execução de um modelo desenhado com base no formalismo de modelação DS-Pnet é realizada usando a semântica passo-máximo de execução (*maximal-step execution*), isto é, todas as transições da rede de Petri habilitadas num determinado passo de execução disparam nesse

mesmo passo; por sua vez, as operações de fluxo de dados são calculadas instantaneamente, sem que ocorra a propagação de atrasos [31].

Estes modelos podem ser desenhados no editor do ambiente IOPT-Flow Editor, um ambiente de desenvolvimento que remete para uma modelação fácil e *user-friendly* e que permite testar e validar implementações efetuadas através de condições especificadas pelo projetista. Efetivamente, IOPT-Flow Editor foi desenvolvido para suportar o projeto de vários controladores de sistemas embutidos e ciberfísicos que combinam elementos de redes de Petri e *dataflow* [31].

O ambiente IOPT-Flow é composto por um conjunto de ferramentas acedidas pela *web* em http://gres.uninova.pt/~clo/iopt-flow/, nomeadamente, um editor gráfico, um simulador local e remoto, e geradores automáticos de código para implementações em JavaScript, VHDL (*VHSIC Hardware Description Language*) e C [1] – Figura 2.8. A combinação do simulador com ferramentas de validação de modelo possibilitará a deteção de erros numa fase precoce da implementação, tais como *deadlocks*, *live locks* e o acesso (*reachability*) a estados de erros [30].

Para além disso, na fase de edição do modelo, é expectável que a rede de Petri consiga lidar com problemas de paralelismo, de concorrência e de sincronismo, tornando a deteção de erros mais simples; do mesmo modo, a utilização de elementos *dataflow* permite especificar operações e dependências entre sinais [30].

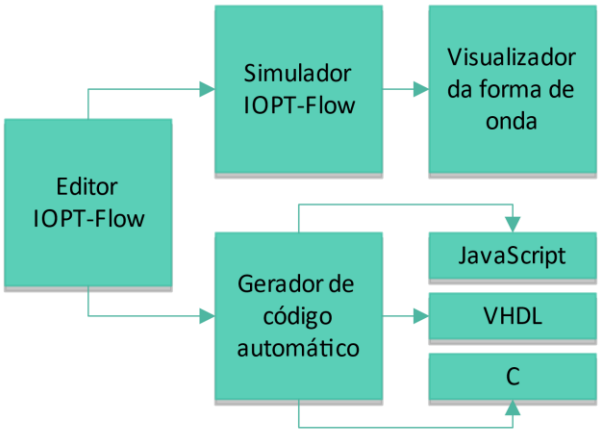


Figura 2.8: Diagrama que relaciona as ferramentas oferecidas pelo ambiente IOPT-Flow Editor.

O editor, simulador e a ferramenta para a visualização da forma de onda foram implementados em PHP (*Hypertext Preprocessor*) e JavaScript e os geradores de código automáticos foram implementados usando XSLT (*eXtensible Stylesheet Language for Transformation*); os ficheiros dos modelos e os componentes das bibliotecas são guardados em ficheiros XML, que podem ser processados por outras linguagens de programação [31].

No final, um modelo projetado pode ser usado como parte de outro projeto, desenvolvido com outras ferramentas e plataformas, devido à facilidade de integração assegurada pela geração de código automático; uma vantagem de portabilidade [30].

2.9.Mercado mundial de cadeiras de rodas elétricas

O mercado de cadeiras de rodas elétricas apresenta-se emergente. O aumento do envelhecimento da população e a ocorrência de lesões físicas causadas por doenças e traumas, por vezes relacionadas com o modo de vida e meio envolvente, contribuem para o crescimento do mercado, que é reforçado pela procura por parte do setor desportivo e influenciado pela demanda de novas tecnologias.

Globalmente, o mercado é segmentado considerando-se o tipo de produto e a geografia no mundo. Efetivamente, identificam-se vários tipos de cadeiras de rodas elétricas: cadeiras de tração dianteira, tração central, tração traseira, de verticalização, entre outros; e regiões onde o mercado se destaca: Ásia-Pacífico, Médio Oriente, África, Europa Ocidental, América Latina e América do Norte [32]. Assim, é expectável que exista uma oferta diversificada, caracterizada por cadeiras com diferentes sistemas e funcionalidades.

Algumas das maiores empresas no mercado de cadeiras de rodas elétricas são: Levo AG, Permobil AB, Pride Mobility Products Co., Hoveround Corporation, Sunrise Medical Ltd., Invacare Co., Medical Depot Inc., Meyra GmbH., OttoBock Healthcare GmbH. e GF Health Products Inc.

Consequentemente, qualquer tecnologia presente no mercado é previamente validada para a verificação de possíveis erros e falhas; de seguida, apresenta-se um conjunto de falhas identificadas que podem ocorrer aquando da utilização de uma cadeira de rodas elétricas: (1) falha no acionamento dos travões; (2) falha na bateria; (3) falha no motor; (4) falha no sistema de iluminação; (5) falha no sistema de comando; (6) danos no joystick e falha no sistema de controlo; (7) falha nos sensores de limitação da velocidade; (8) falha de rede ou na configuração; (9) falha no dispositivo de posicionamento.

Habitualmente, a pesquisa e o trabalho desenvolvido no âmbito do estudo de cadeiras de rodas elétricas assenta no melhoramento do sistema de controlo, nomeadamente a sua personalização considerando a patologia do utilizador e a sua capacidade de interação com o sistema; na avaliação da melhor trajetória ou análise do ambiente envolvente e dos obstáculos detetados, aumentando a capacidade de mobilidade do utilizador; no projeto de soluções que otimizam o funcionamento dos motores, da bateria e do custo monetário da cadeira; e na necessidade de monitorização dos sinais vitais do utilizador para efeitos de tratamento médico. Analogamente, é possível idealizar-se um sistema monitor dos parâmetros associados ao funcionamento da cadeira; os objetivos serão idênticos, permitindo também facilitar a deteção de erros e falhas no sistema, acelerando a respetiva resolução por parte dos técnicos.

A Invacare Co. é líder mundial na produção e distribuição de produtos que promovem a recuperação médica e o estilo de vida dos clientes. Em setembro de 2017, a empresa apresentou a ferramenta digital MyLiNX app. MyLiNX app permite que o utilizador da cadeira tenha acesso a informação clara sobre o estado da cadeira, nomeadamente o estado da bateria, dos motores e códigos de ocorrência de falha, apoiando o diagnóstico; possibilita o descarregamento de toda a

informação atual do sistema e diagnóstico, suportando a comunicação entre a cadeira e o fornecedor; fornece dados sobre as falhas à equipa de suporte, que pode eliminar a necessidade da intervenção de um técnico.

Assim, a ocorrência de erros e falhas pode ser identificada através da aplicação; esta apresenta o código do erro ocorrido, o módulo que foi afetado e uma descrição que sugere uma resolução. Para além disso, o histórico armazena a ocorrência de erros cronologicamente, sendo possível identificar-se uma tendência na ocorrência de um erro, e grava as estatísticas de como a cadeira tem sido usada, enquanto a informação da performance pode ser visualizada em tempo real [33].

Finalizando o capítulo, tudo o que foi exposto imediatamente acima, apoia a proposta do desenvolvimento do ambiente IOPT-Flow GUI Editor.

O objetivo é disponibilizar um ambiente onde o utilizador possa editar – e programar visualmente – uma interface gráfica de utilizador por meio do arrasto dos elementos que compõem a interface, mais adiante designados *widgets*, que serão caracterizados por um conjunto de *tags*, que definem as suas características na página HTML do ambiente. O ambiente terá características semelhantes às ferramentas apresentadas anteriormente: possibilitará a posição dos elementos na área de desenho, como acontece com o CUDABlock ou o LabVIEW; e a caracterização do parâmetros e atributos dos elementos, como acontece com o ambiente Visual Studio ou WindowBuilder.

Para além disso, tal como *Animator* e o *Synoptic*, o ambiente IOPT-Flow GUI Editor permite relacionar a GUI de um sistema com o respetivo modelo comportamental, neste caso modelado segundo o formalismo de modelação DS-Pnet. Para que tal seja possível, a interface criada pode ser representada no formato XML, que corresponde a geração automática de componentes IOPT-Flow para o ambiente IOPT-Flow Editor, garantida pelo novo ambiente.

No capítulo que se segue, detalha-se a relação do novo ambiente com o ambiente IOPT-Flow Editor; e é apresentada a estrutura do novo ambiente, as suas características, as ferramentas que oferece para a edição e geração automática de GUI.

3

Proposta e Ambiente Desenvolvido

O ambiente IOPT-Flow GUI Editor – adiante, também designado de GUI Editor – possibilita a construção de interfaces gráficas de utilizador, permitindo ao *designer* posicionar *widgets* visualmente com suporte ao *drag-and-drop*, e definir parâmetros respeitantes às propriedades e características dos mesmos. A qualquer momento, também é possível gerar o modelo e o componente IOPT-Flow da interface, que podem ser associados a outros modelos e componentes desenvolvidos no ambiente IOPT-Flow Editor, cujos comportamentos dependem da interação com um utilizador; cada *widget* da interface é convertido para o componente IOPT-Flow correspondente, e o conjunto de todos esses componentes criam o modelo e o componente IOPT-Flow da interface.

No presente capítulo são descritos o editor e o gerador de interfaces gráficas de utilizador desenvolvidos neste trabalho. Primeiro, é explicado de que forma é que o ambiente desenvolvido neste trabalho se relaciona com o ambiente IOPT-Flow Editor. Depois, é apresentado um diagrama de caso de uso do ambiente proposto, seguido da apresentação da sua área de trabalho e de cada uma das três áreas de interação com o utilizador: menu ou *toolbox*, área de desenho e área de exibição e configuração das propriedades e características dos elementos de uma GUI. Posteriormente, é descrito como são gerados o modelo e o componente IOPT-Flow de uma interface, e como estão armazenados os ficheiros e os recursos associados ao seu desenvolvimento.

O novo ambiente, disponível em http://gres.uninova.pt/~clo/iopt-flow/inter_clo.php, está instalado num servidor Apache HTTP (*Hypertext Transfer Protocol*) em execução no Linux. Para a implementação do IOPT-Flow GUI Editor foi utilizado a versão 68.0.3440.106 do navegador Google Chrome;.o ambiente WinSCP (Windows Secure CoPy) que usa o protocolo SFTP (SSH File Transfer Protocol) e permite aceder ao servidor onde está alojado o novo ambiente; o editor Visual Studio Code, usado para a edição de ficheiros de código – neste caso, JavaScript, HTML e PHP; e o ambiente IOPT-Flow Editor, utilizado para validar resultados intermédios.

3.1. Integração entre os ambientes IOPT-Flow GUI Editor e IOPT-Flow Editor

Para inicializar o desenvolvimento da ferramenta proposta neste trabalho, partiu-se do IOPT-Flow Editor que passou a fornecer uma ligação ao GUI Editor – através do evento *onclick* de um botão que se situa no primeiro ambiente, dá-se a abertura de uma nova janela ou separador direcionado para o novo ambiente de edição e geração automática de interfaces gráficas de utilizador. Desta forma, realizou-se uma primeira fase de integração entre os dois ambientes. A relação entre eles está esquematizada na Figura 3.1.

Um dos principais objetivos deste trabalho é a criação de interfaces gráficas de utilizador para serem abertos e usados no ambiente IOPT-Flow Editor, e facilmente interligados com outros modelos e componentes DS-Pnet.

Como os modelos e componentes DS-Pnet são armazenados em documentos XML [29] e o IOPT-Flow Editor trabalha diretamente nesses documentos, a nova ferramenta, para além de permitir a edição de GUI, faz a sua tradução – e dos seus *widgets* – para um modelo e componente IOPT-Flow, através da geração automática de dois documentos XML.

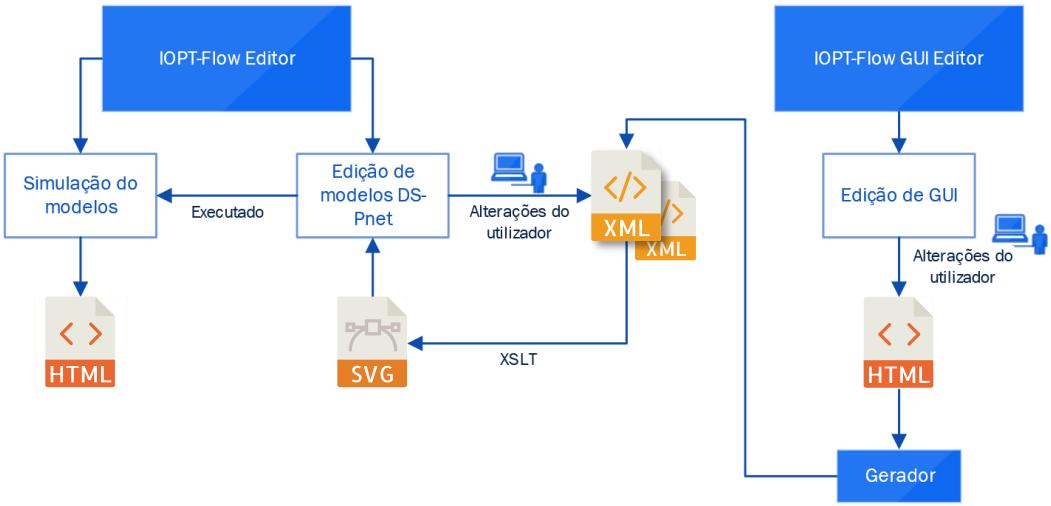


Figura 3.1: Relação entre o ambiente desenvolvido e o ambiente IOPT-Flow Editor.

Assim, no IOPT-Flow GUI Editor podemos criar a interface de um sistema através da edição de um documento HTML em tempo real, adicionando *widgets* com propriedades e características que podem ser personalizadas considerando os requisitos de modelação do sistema em causa; e gerar os ficheiros XML correspondentes, que representam o modelo e o componente IOPT-Flow da interface criada, possibilitando a sua visualização e utilização no ambiente IOPT-Flow Editor: a interface passa a poder ser usada neste ambiente, onde apenas é necessário estabelecer a sua ligação a um modelo que possa modelar o seu comportamento, garantindo o seu bom funcionamento.

A Figura 3.2 apresenta o exemplo de um modelo DS-Pnet desenhado no ambiente IOPT-Flow Editor, e o respetivo componente. O modelo é composto por uma entrada, duas saídas e uma constante com valor igual a 5; o componente apresenta-se como uma caixa negra, onde apenas se visualizam as entradas e saídas do modelo.

(a) Modelo (b) Componente

Figura 3.2: Modelo e respetivo componente IOPT-Flow.

Para a visualização do modelo ou do componente no IOPT-Flow Editor, é usado SVG (Scalable Vector Graphics), obtido pela transformação XSLT do documento XML, o que permite observar a representação gráfica dos ficheiros XML gerados, ou seja, a conversão da GUI num modelo ou componente IOPT-Flow equivalente.

É expectável que a interface que se elabora no novo GUI Editor seja a que posteriormente o utilizador possa observar quando a acrescenta a um projeto no IOPT-Flow Editor e o simula. O que acontece é que o documento HTML gerado aquando da simulação em IOPT-Flow, é gerado a partir do documento XML aberto nesse ambiente, e não é o mesmo HTML que o utilizador edita no novo ambiente para criar a interface. Assim, foi necessário garantir a correta geração dos ficheiros XML.

Garantindo que os dois documentos HTML são os mais idênticos possível, ajudamos o utilizador a ter a perceção correta de como será a interface que o seu projeto apresentará durante uma simulação ou execução no IOPT-Flow Editor. Para além disso, conseguimos garantir que os dois ambientes – IOPT-Flow Editor e IOPT-Flow GUI Editor – são interoperáveis e que os ficheiros XML gerados correspondem ao verdadeiro modelo e componente IOPT-Flow da GUI editada.

No subcapítulo que se segue, apresenta-se o diagrama de caso de uso do novo ambiente.

3.2. Diagrama caso de uso do ambiente IOPT-Flow GUI Editor

O diagrama caso de uso apresentado na Figura 3.3 descreve o funcionamento do ambiente desenvolvido neste trabalho, nomeadamente a interação que um utilizador e *designer* de GUI pode ter com o ambiente IOPT-Flow GUI Editor.

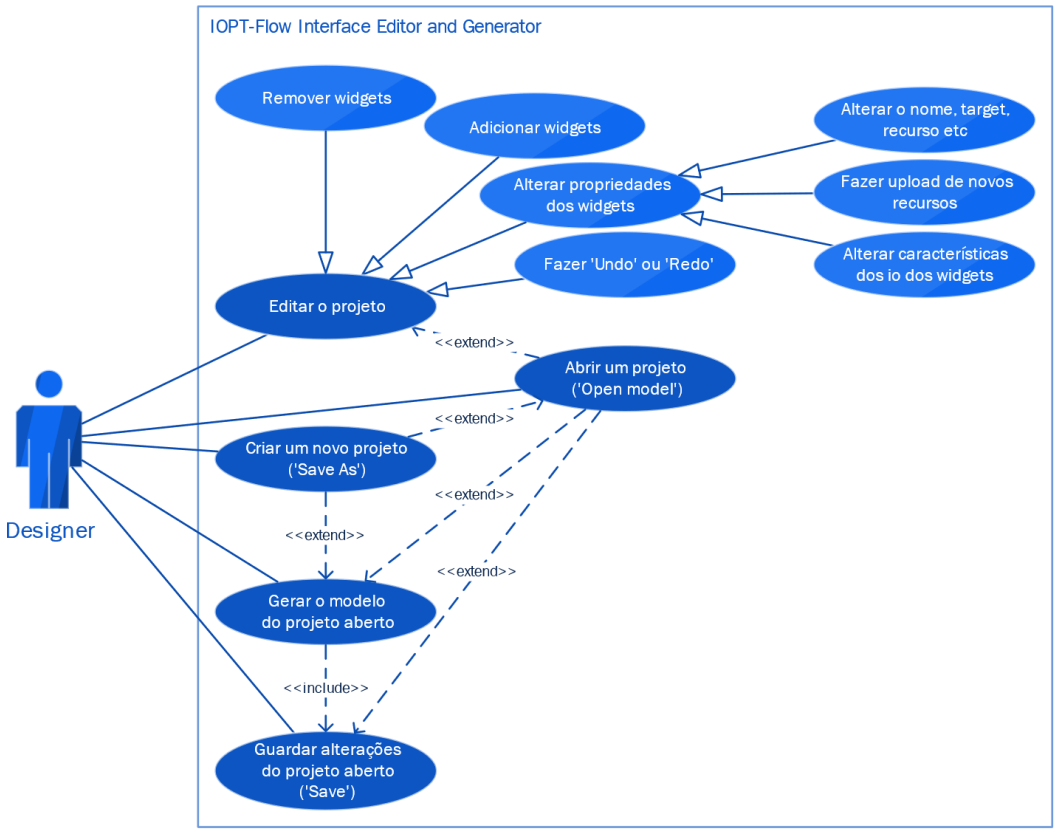


Figura 3.3: Diagrama caso uso do ambiente IOPT-Flow GUI Editor.

Existem algumas condições associadas a cada uma das ações, estando esquematizadas na Figura 3.3 pelo desenho das relações *extend*, *include* e generalização. As condições respeitantes às relações *extend* e *include* são as seguintes:

- O utilizador pode abrir um projeto existente para o continuar a editar, ou começar a editar uma nova interface;
- Para o utilizador abrir um projeto, ele tem de existir; se não existir, ele pode a qualquer momento criar um;
- O utilizador só pode guardar alterações de um projeto caso este exista e esteja aberto; caso não exista, para guardar as alterações de uma interface que esteja a ser editada, é aconselhável a criação de um novo projeto, sendo as alterações automaticamente guardadas;
- O utilizador só pode gerar o modelo e o componente IOPT-Flow de um projeto que exista e esteja aberto; se for o caso, guarda as últimas alterações feitas ao projeto e gera os ficheiros XML da tradução do modelo GUI para IOPT-Flow; caso o projeto não exista, o utilizador vai ser inquirido no sentido de o criar, e se aceitar, a interface editada é guardada e são gerados os ficheiros XML com a tradução para um modelo e componente IOPT-Flow.

Aquando da edição de um modelo GUI, o utilizador pode adicionar e remover *widgets*, organizando e compondo a sua interface com recurso ao *drag-and-drop* e às funções *redo*, *undo* e

delete. Por forma a facilitar a integração da interface com o sistema para o qual está a ser desenvolvida, o utilizador pode personalizar as propriedades de cada *widget*, como por exemplo, o nome, o *target*, o recurso; gerir recursos, nomeadamente imagens e sons usados para serem apresentadoss na GUI, sendo a gestão global a todo o projeto e não só referente a um *widget*; e definir características sobre as entradas e saídas que estarão associadas ao *widget* no modelo e componente IOPT-Flow.

Nos próximos subcapítulos, volta-se a explicar com mais detalhe o ambiente desenvolvido, incluindo também uma explicação sobre a organização que foi definida para guardar os modelos GUI e DS-Pnet e os conteúdos de imagem usados na elaboração de um projeto.

3.3. Ambiente IOPT-Flow GUI Editor

No trabalho desenvolvido no âmbito desta dissertação foi usada uma abordagem de organização do ambiente idêntica à do IOPT-Flow Editor. O novo ambiente apresenta três áreas de interação com o utilizador: à esquerda, um menu, ou *toolbox*, que oferece várias ferramentas de desenvolvimento; à direita, uma área onde são apresentadas as propriedades e características de um *widget* que seja selecionado – se não for selecionado qualquer *widget*, a área é omissa; e ao centro, uma área de edição e desenho, onde o utilizador pode compor a sua interface, e definir como deseja que esta se relacione com um sistema. O ambiente IOPT-Flow GUI Editor é o que se apresenta na Figura 3.4.

Figura 3.4: Ambiente IOPT-Flow GUI Editor.

No desenvolvimento do IOPT-Flow GUI Editor foi usada uma combinação de tecnologias Web; para desenvolver a parte interativa do editor foram escolhidas as linguagens de programação HTML e JavaScript e o modelo de representação DOM (*Document Object Model*); e para aceder ao servidor e executar tarefas relacionadas com o tratamento de ficheiros e recursos dos projetos, foi usado PHP.

Todas as ações realizadas pelo utilizador no editor correspondem a manipulações dos elementos da página HTML do ambiente. Estas manipulações são realizadas por funções JavaScript apoiadas pelo DOM, que promove uma representação da página, ou documento, como um conjunto estruturado de nós e objetos com propriedades, métodos e eventos.

Efetivamente, neste trabalho foram implementadas funções em JavaScript e PHP, invocadas por eventos dos elementos que são manipulados pelo utilizador, e que permitem recuperar modificar, atualizar e remover conteúdos da página HTML do ambiente. Tome-se como exemplo o posicionamento dos *widgets* na área de desenho; esta ação é conhecida por *drag-and-drop*, caracterizada pela deteção de eventos que traduzem o estado do rato: *onmousemove*, que ocorre quando se deteta que o rato está em movimento; *onmousedown*, que ocorre quando se deteta que o botão do rato está a ser pressionado; e *onmouseup*, que ocorre quando se deteta que o botão do rato deixa de ser pressionado.

Mais adiante será apresentada a função implementada para realizar o *drag-and-drop*. No subcapítulo seguinte apresenta-se a área *toolbox* do editor e destacam os *widgets* disponíveis.

3.3.1. Menu

Ao aceder ao IOPT-Flow GUI Editor, a primeira ação que o utilizador terá com o novo ambiente, será necessariamente com a área *toolbox*. A Figura 3.5 apresenta a área *toolbox*, que oferece várias ferramentas dedicadas à edição e geração automática de interfaces gráficas de utilizador.

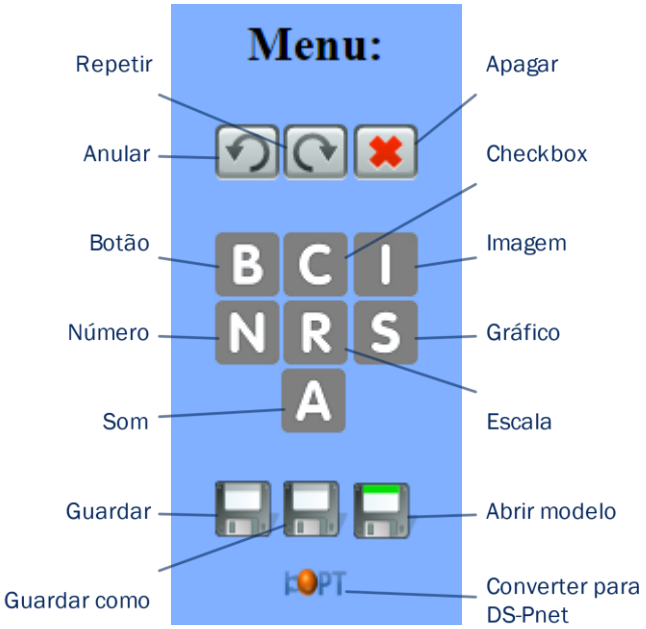


Figura 3.5: Menu do ambiente IOPT-Flow GUI Editor.

Seguidamente, a Figura 3.5 será analisada de cima para baixo, da esquerda para a direita.

No menu do IOPT-Flow GUI Editor existem três botões dedicados ao controlo das ações do utilizador na área de desenho; os botões 'Anular', 'Repetir' e 'Apagar' são usados para retroceder,

ou alterar edições não intencionadas no desenho de uma GUI. Como o nome indica, 'Anular' serve para anular ações; 'Repetir' serve para reproduzir ações – caso tenham sido anuladas; e 'Apagar' serve para eliminar qualquer elemento, ou *widget*, que tenha sido adicionado ao desenho de uma interface.

Ao centro do menu, a meio da Figura 3.5, encontram-se o conjunto de *widgets* que são oferecidos para a edição e o desenho de uma GUI. Os botões 'Botão', 'Checkbox', 'Imagem', 'Número', 'Escala', 'Gráfico' e 'Som', correspondem a *widgets* do tipo botão, *checkbox* ou caixa de seleção, imagem, número, escala ou barra graduada, gráfico e som ou áudio, respetivamente. Posteriormente é dado foco a cada um destes elementos.

Por fim, existe um grupo de botões implementados para a apoiar a gestão de ficheiros relacionados com um projeto. Através dos botões 'Guardar', 'Guardar como' e 'Abrir modelo' é possível atualizar, criar e abrir ficheiros HTML de projetos desenvolvidos no novo ambiente, respetivamente; e guardar ou atualizar os recursos usados nesses projetos. Por outro lado, a criação ou atualização do modelo e componente DS-Pnet de uma GUI é realizada através do botão 'Converter para DS-Pnet', que cria dois ficheiros XML com a sua tradução para DS-Pnet e guarda-os, ou atualiza as versões que possam existir.

De seguida, são apresentados os *widgets* oferecidos pelo ambiente IOPT-Flow GUI Editor.

3.3.1.1. Widgets

Previamente, foi explicado como é que o ambiente desenvolvido nesta dissertação se relaciona com o ambiente IOPT-Flow Editor. Foi referido que qualquer GUI que se elabore no novo ambiente, deverá ser a que o utilizador observa, quando a acrescenta a um projeto no IOPT-Flow Editor e o executa; e que isso é garantido pela correta criação do modelo e componente IOPT-Flow da GUI no novo ambiente.

O ambiente IOPT-Flow Editor disponibiliza componentes que podem ser usados na criação de interfaces gráficas de utilizador, – encontram-se disponíveis na pasta "ui" da biblioteca. Aquando do desenvolvimento da primeira versão do IOPT-Flow GUI Editor, foram implementados um conjunto de *widgets*, os quais se sabia serem também disponibilizados pelo IOPT-Flow Editor, e, portanto, reconhecidos por ele após a tradução de uma GUI para IOPT-Flow.

A Figura 3.6 apresenta os *widgets* implementados, pela seguinte ordem: botão, *checkbox* ou caixa de seleção, imagem, número, escala ou barra graduada, gráfico e som ou áudio.

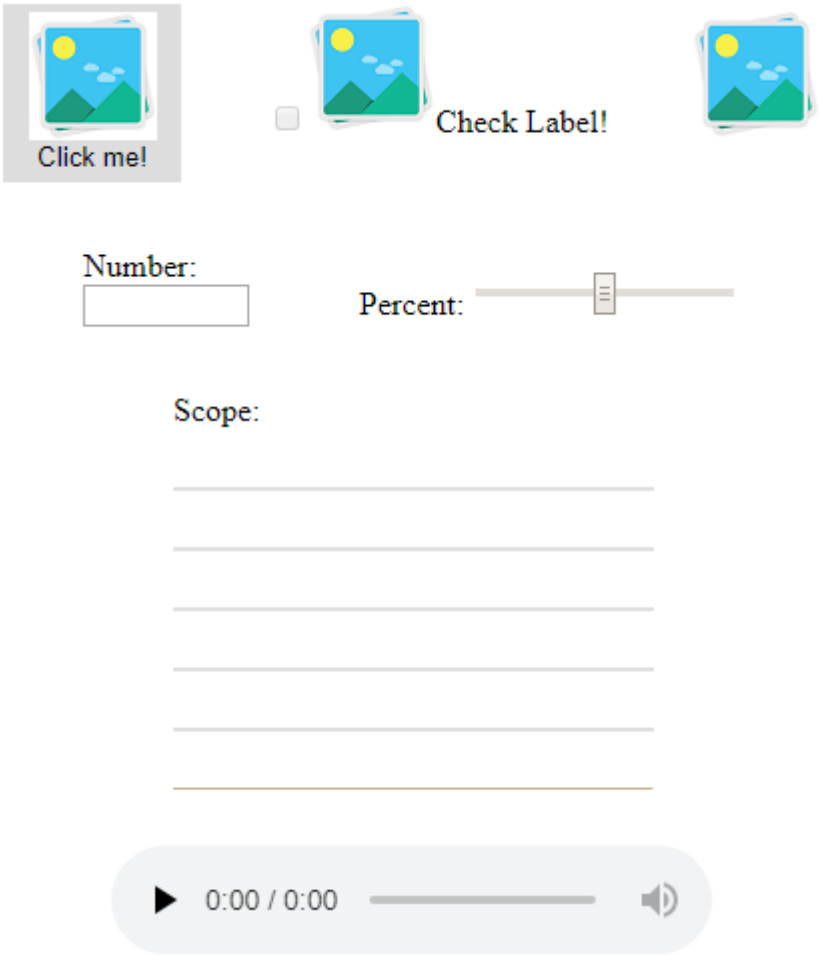


Figura 3.6: *Widgets* usados no desenho de interfaces gráficas de utilizador no ambiente IOPT-Flow GUI Editor.

A Figura 3.6 apresenta os o *widgets* que aparecem na área de edição, quando são adicionados ao desenho de uma interface. Por padrão, o botão e a *checkbox* apresentam uma imagem e uma legenda, que podem ser modificadas no painel das propriedades, apresentado neste capítulo; também a imagem, referente ao *widget* com o mesmo nome, pode ser alterada.

Na verdade, todos os *widgets* possuem um conjunto de propriedades e características que, sendo importantes para a geração do seu componente e, por conseguinte, do modelo e componente IOPT-Flow de uma GUI, foram associados ao seu elemento HTML através de atributos e outros elementos. Assim, o utilizador pode personalizar os *widgets* através da manipulação – indireta – dos seus elementos HTML.

O excerto seguinte – Figura 3.7 – apresenta parte do código que permite a criação do elemento botão no novo ambiente.

```
var button = document.createElement("button");

button.setAttribute('id', nextId("button"));
button.setAttribute('name', 'ui_button_' + button.getAttribute('id'));
button.setAttribute('classxml', 'ui/button.xml');
button.setAttribute('implementation', 'iopt-flow');
button.setAttribute('target', 'external');
button.setAttribute('resourcelocation', base_dir+'padrao.png');
button.setAttribute('param_string', '@ @');
button.setAttribute('comment', 'Click me!');
button.setAttribute('in_out', 'I Visible: boolean\nI Sensitive: boolean\nI X: range[0:1023]\
nI Y: range[0:1023]\nI Width: range[0:1023]\nI Height: range[0:1023]\nI PageNr: range[0:31]\
nO Click: event\nO Pressed: boolean');

for(var io = 0; io < io_button.length; ++io)
{
      var inout = document.createElement("var");
      inout.setAttribute('id', io_button[io]);
      inout.setAttribute('opt', 0);
      for(var a = 0; a < attributes.length; ++a)
      {
            if(io_button[io] != 'click') inout.setAttribute(attributes[a], " ");
            else if(a!=2 && a!=3 && a!=4) inout.setAttribute(attributes[a], " ");
      }
      button.appendChild(inout);
}
```

Figura 3.7: Excerto do código que permite a criação do elemento do tipo botão no ambiente IOPT-Flow GUI Editor.

Como se pode observar na Figura 3.7, quando se cria um botão – ou qualquer outro *widget* – é definido um conjunto atributos através do método setAttribute(). O conjunto corresponde às propriedades do *widget*: "id", "name", "classxml", "implementation", "target", "resourcelocation", "param_string", "comment" e "in_out"; e serão analisados no subcapítulo onde é apresentada a área das propriedades e características dos *widgets*.

Do mesmo modo, destaca-se o ciclo "for", que a cada iteração cria e atribui ao botão uma variável, um elemento do tipo "var", que corresponde a uma determinada entrada ou saída do componente IOPT-Flow do *widget*. Por outras palavras, o array "io_button" é composto pelos nomes das entradas e saídas do componente IOPT-Flow do tipo botão: "visible", "sensitive", "x", "y", "width", "height", "click" e "pressed". Cada entrada e saída é o *id* de um "var", ao qual também se atribuem um conjunto de atributos: "name", "mode", "type", "min", "max" e "value". No caso da saída "click", que corresponde à ocorrência de um evento *onclick* no botão, não foram atribuídos os atributos "type", "min" e "max"; tal acontece a todos as entradas e saídas que são do tipo evento; veja-se a Tabela 2.1 e a Tabela 3.1. Isto deve-se a facto de um evento não possuir tipo de valores e um intervalo onde possam variar.

Contudo, como entradas e as saídas variam de componente para componente, a atribuição dos elementos "var" ao elemento HTML de um *widget*, depende de *widget* para *widget*. Atente-se à Figura 3.8, que apresenta os componentes IOPT-Flow de cada um dos *widgets* implementados – os componentes estão na ordem usada para apresentar os *widgets* na Figura 3.6.

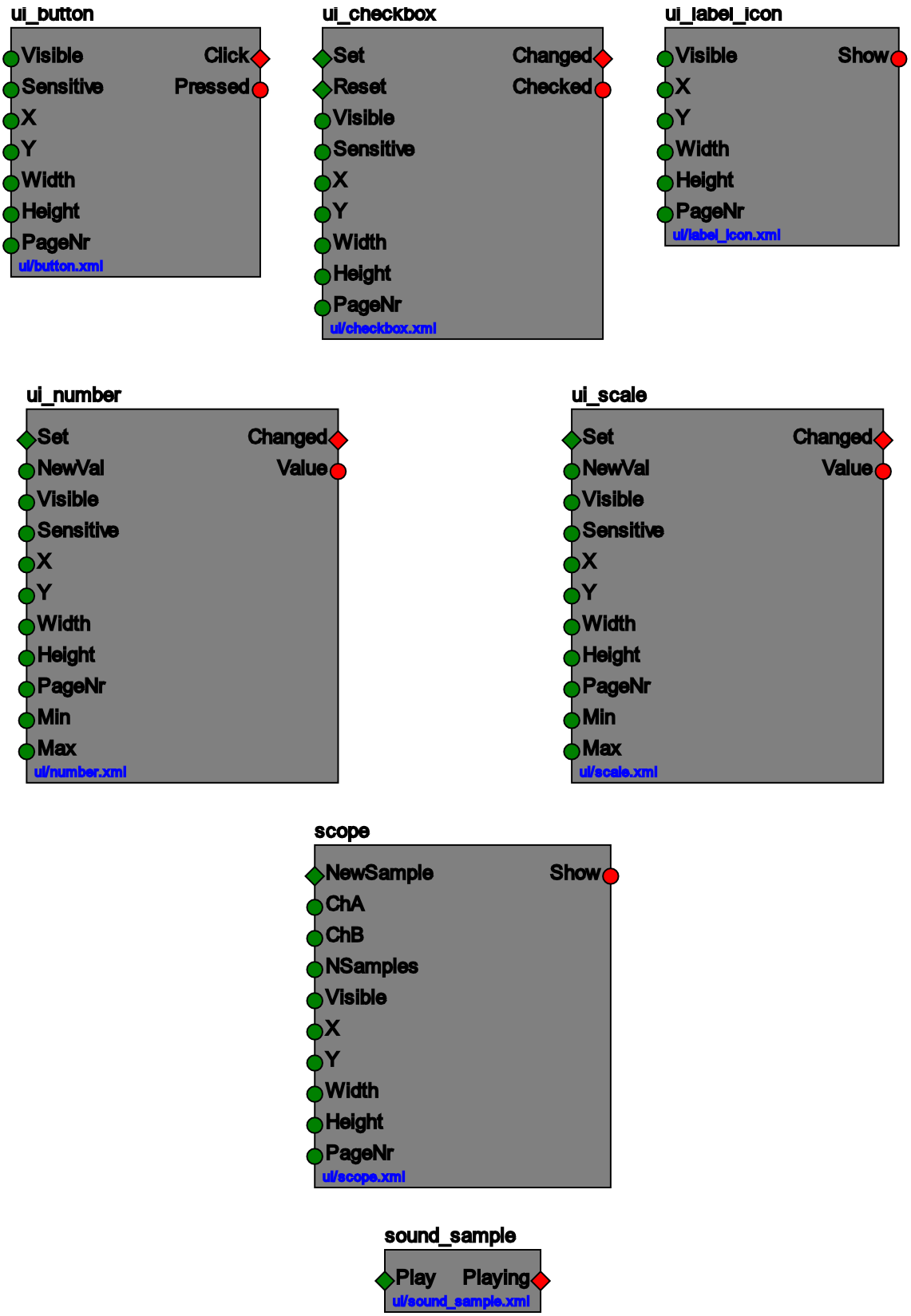


Figura 3.8: Componentes IOPT-Flow usados no desenho de interfaces gráficas de utilizador no ambiente IOPT-Flow Editor, e gerados automaticamente pelo ambiente IOPT-Flow GUI Editor.

Assim, a representação de um botão no modelo DOM da página HTML do novo ambiente, após a sua adição a uma GUI, é a que se apresenta na Figura 3.9.

```
<button class="drag" id="b001" name="ui_button_b001" classxml="ui/button.xml"
implementation="iopt-flow" target="external" resourcelocation="./tmp/padrao.png"
param_string="@ @" comment="Click me!" in_out="I Visible: boolean
I Sensitive: boolean
I X: range[0:1023]
I Y: range[0:1023]
I Width: range[0:1023]
I Height: range[0:1023]
I PageNr: range[0:31]
O Click: event
O Pressed: boolean" style="border-width: 3px; border-style: solid; border-color: rgb(255,
255, 255); padding: 0px; position: absolute; visibility: visible; left: 0px; top: 0px; width:
95px; height: 95px;">
      <var id="visible" opt="0" name=" " mode=" " type=" " min=" " max=" " value=" "></var>
      <var id="sensitive" opt="0" name=" " mode=" " type=" " min=" " max=" " value=" "></var>
      <var id="x" opt="0" name=" " mode=" " type=" " min=" " max=" " value=" "></var>
      <var id="y" opt="0" name=" " mode=" " type=" " min=" " max=" " value=" "></var>
      <var id="width" opt="0" name=" " mode=" " type=" " min=" " max=" " value=" "></var>
      <var id="height" opt="0" name=" " mode=" " type=" " min=" " max=" " value=" "></var>
      <var id="out" opt="0" name=" " mode=" " type=" " min=" " max=" " value=" "></var>
      <var id="event_out" opt="0" name=" " mode=" " type=" " min=" " max=" " value=" "></var>
      <img src="./tmp/padrao.png" style="border: 0px;">
      <p style="border: 0px; margin: 1px;">Click me!</p>
</button>
```

Figura 3.9: Excerto do código que permite a visualização do elemento botão na página HTML do ambiente IOPT-Flow GUI Editor.

Na Figura 3.9, para além dos parâmetros referidos anteriormente, também se podem observar os parâmetros que definem o estilo do *widget* na interface, através do atributo "style", os elementos "img" e "p", respeitantes à imagem e à legenda do *widget*, e a classe que lhe é atribuída; todos os *widgets* pertencem à classe "drag", necessária para a implementação do *drag-and-drop* dos elementos na área de desenho. As representações dos elementos dos restantes *widgets* são idênticas, seguindo a estrutura da Figura 3.9.

No subcapítulo que se segue, é apresentada a área de desenho e edição de uma GUI no ambiente IOPT-Flow GUI Editor.

3.3.2. Área de desenho

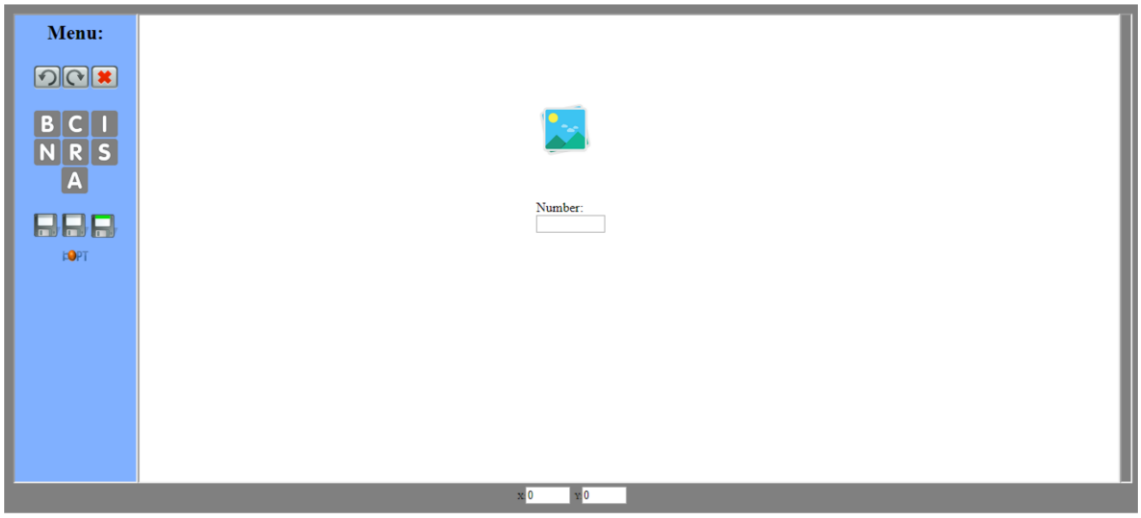
A área de desenho do IOPT-Flow GUI Editor é a área de dimensão predominante, encontrando-se ao centro do editor. Esta área é dedicada ao desenho de interfaces gráficas de utilizador, sendo a região onde podem ser adicionados – ou removidos – e dispostos os *widgets* disponibilizados no menu, e apresentados no subcapítulo anterior.

Aquando da elaboração de uma GUI, o seu desenho é auxiliado pela propriedade *drag-and-drop*, implementada através da deteção de eventos que traduzem o estado do rato:

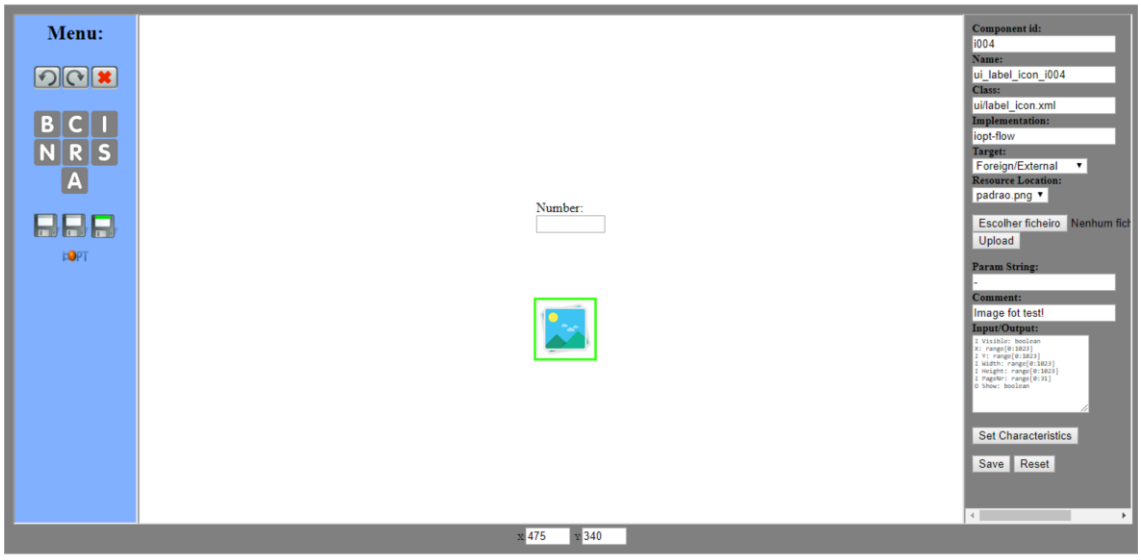
- O evento *onmousedown* é usado para detetar a ocorrência de um clique, ou pressão no botão do rato sobre a área de desenho; quando ocorre, analisa-se se existe algum *widget*, neste caso, pertencente à classe "drag", sobre o ponteiro do rato, através do método closest();

- Durante o movimento do rato, detetado através do evento *onmousemove*, é analisada a nova posição do *widget* na área de desenho, considerando-se sempre os limites que definem a área e a dimensão do elemento; o *widget* acompanha o movimento do rato;
- Quando é detetada a ocorrência dos eventos *onmouseup* ou *onmouseout*, que correspondem à libertação do botão do rato e ao deslocamento do rato para fora da área de desenho, respetivamente, o valor da posição do *widget* na interface é atualizado, e este permanece no último lugar definido pelo ponteiro.

A Figura 3.10 apresenta o que acontece quando a posição de um elemento numa GUI é alterada. Em (a) observamos uma GUI com um *widget* do tipo imagem, e outro do tipo número, e a suas posições iniciais; em (b) observamos o deslocamento vertical da imagem, que anteriormente estava acima do número, e agora encontra-se abaixo.



(a)



(b)

Figura 3.10: Exemplo da interação do utilizador com a área de desenho do ambiente IOPT-Flow GUI Editor, onde são apresentados dois *widgets*.

Como se pode observar, a imagem foi deslocada para uma posição final, diferente da inicial, e ficou destacada na GUI aquando da interação do rato; para além disso, o painel lateral direito tornou-se visível, apresentando as propriedades e características do *widget* selecionado.

No subcapítulo seguinte, é apresentada a área do IOPT-Flow GUI Editor que exibe as propriedades e características de um *widget*.

3.3.3. Área de apresentação e configuração das propriedades e características dos widgets

A terceira área de interação do utilizador com o ambiente IOPT-Flow GUI Editor é a que se apresenta neste subcapítulo: área de apresentação e configuração das propriedades e características dos widgets – Figura 3.11.

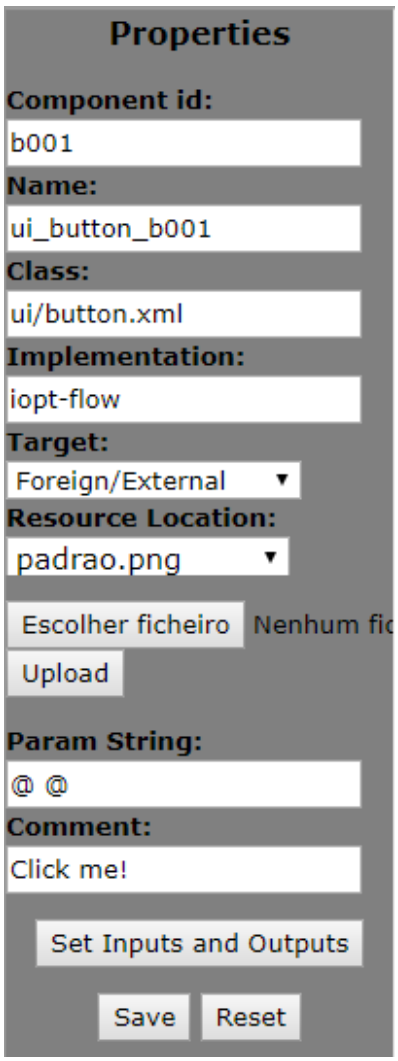


Figura 3.11: Painel que apresenta as propriedades de um *widget* do tipo botão, que foi selecionado na área de desenho do ambiente IOPT-Flow GUI Editor.

Como referido anteriormente, o IOPT-Flow Editor oferece um conjunto de componentes que podem ser usados na implementação de modelos DS-Pnet, inclusive no desenho de interfaces gráficas de utilizador – Figura 3.8.

Quando se cria um modelo DS-Pnet no ambiente IOPT-Flow Editor, todos os componentes adicionados evidenciam um conjunto de propriedades que caracterizam o seu funcionamento e, como tal, podem ser editadas afim de cumprir requisitos do sistema ao qual pertencem. Por exemplo, os componentes usados na elaboração de uma GUI apresentam o seguinte conjunto de propriedades: um identificador, um nome, uma classe, um recurso, um parâmetro e um comentário, e informação sobre o tipo de implementação, o *target* e as entradas e saídas do componente.

Como um dos principais objetivos deste trabalho é a criação automática de modelos e componentes DS-Pnet de interfaces gráficas de utilizador, o novo ambiente, IOPT-Flow GUI Editor, dispõe de uma área dedicada à edição das propriedades dos *widgets*, apoiando o desenvolvimento de uma GUI e a correta conceção do modelo e do componente DS-Pnet da mesma, para posterior utilização no IOPT-Flow Editor.

A Figura 3.11 apresenta a área que exibe as propriedades de um *widget*, no novo ambiente. Assim, tal como os componentes no IOPT-Flow Editor, no IOPT-Flow GUI Editor cada widget é caracterizado por um identificador, um nome, uma classe, um recurso, um parâmetro e um comentário – os três últimos são opcionais – e informação sobre o tipo de implementação e o *hardware* a que se destina; estes parâmetros podem ser todos editados no painel da Figura 3.11. Para além disso, o painel apresenta uma área que pode ser acedida através do botão "Set Inputs and Outputs", e que é dedicada à edição de outras características do *widget*, referentes à atribuição de valores a entradas e saídas do componente DS-Pnet do mesmo – Figura 3.12; estas características são apresentadas neste subcapítulo, mais adiante.

Na Figura 3.11, o campo de cada um dos parâmetros está preenchido com informação sobre um *widget* do tipo botão. De todos os parâmetros, o utilizador não está autorizado a alterar o valor do identificador, da classe e do tipo de implementação. A classe informa sobre a localização do tipo de componente do *widget* no ambiente IOPT-Flow; e o tipo de implementação refere o formalismo usado para desenvolver e especificar o comportamento desse componente.

Relativamente às restantes propriedades, o *target* é a propriedade do componente que informa sobre o gerador de código usado; as opções são: "Default", "Software", "Hardware", "Distributed/Remote" ou "Foreign/External". Esta propriedade apoia soluções *co-design*, onde alguns componentes são implementados como software e outros como hardware.

As propriedades parâmetro e recurso suportam a implementação de sistemas distribuídos: o parâmetro é usado para passar dados, como por exemplo, o valor de uma porta de comunicação ou atalhos do teclado; e o recurso fornece a localização remota de recursos, neste caso, a localização de ficheiros de imagem, para os *widgets* do tipo botão, *checkbox* e imagem, estando esta propriedade ativa apenas para estes *widgets*.

Assim, a propriedade recurso permite ao utilizador selecionar a imagem que quer associar ao *widget* – botão, *checkbox* ou imagem – através de uma lista que lhe é fornecida, e à qual se pode

adicionar mais imagens via *upload* de ficheiros de imagem que se encontrem no computador do utilizador.

Foi referido que, o utilizador pode abrir um projeto existente para o continuar a editar, ou começar a editar uma nova interface no momento em que acede ao ambiente GUI Editor. Ambos os cenários são válidos, caso o utilizador tenha sessão iniciada no ambiente IOPT-Flow editor, ou não. Se o utilizador aceder ao novo ambiente e começar a editar uma interface que ainda não está guardada ou associada a algum projeto, os recursos aos quais terá acesso estarão armazenados numa pasta temporária no servidor – a pasta "tmp"; caso esteja a editar um projeto existente os recursos estarão localizados na pasta do projeto e qualquer upload será armazenado lá. Esta perspetiva de organização dos ficheiros será explicada com mais detalhe no subcapítulo seguinte.

A propriedade comentário é usada para apresentar mensagens de texto na GUI, não sendo válida para os *widgets* do tipo imagem ou som. Na verdade, como o ambiente IOPT-Flow Editor não oferece o uso de componentes do tipo textual, todas as mensagens que são necessárias apresentar numa GUI são definidas com o uso do comentário do respetivo *widget*.

Um outro conjunto de parâmetros que podem ser modificados são os do painel da Figura 3.12, usado para editar valores referentes aos sinais ou eventos de entrada e saída de um componente de um *widget*.

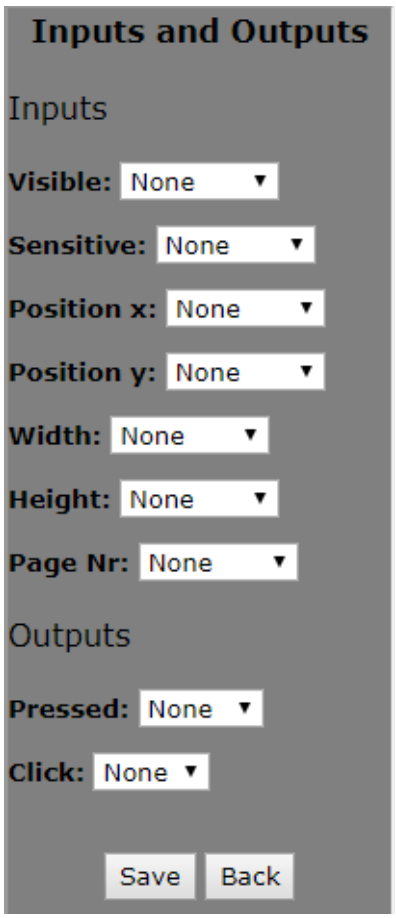


Figura 3.12: Painel que apresenta as entradas e saídas de um *widget* do tipo botão, que foi selecionado na área de desenho do ambiente IOPT-Flow GUI Editor.

Como se pode observar na Figura 3.8, o número e tipo de entradas e saídas de um componente depende da classe a que este pertence – ex.: ui/button.xml. Como tal, no painel da Figura 3.12 são apresentadas apenas as entradas e saídas que correspondem ao componente DS-Pnet do *widget* selecionado.

Imagine-se que no ambiente IOPT-Flow Editor é criado um projeto que modela o funcionamento de um sistema. Se o modelo for desenvolvido com o formalismo DS-Pnet, o utilizador conhece o seu desenho e pode usar o modelo ou o respetivo componente – Figura 3.2. Caso o modelo seja projetado usando outros formalismos de modelação e linguagens de desenvolvimento, pode-se criar um componente contendo apenas os sinais e eventos de saída e entrada, necessários para interagir com o modelo.

Considere-se agora o objetivo deste trabalho. A criação e geração automática de uma interface gráfica de utilizador, que será desenvolvida separada do modelo para o qual se destina. O utilizador deverá perceber como funciona o modelo, quais são os seus requisitos e a sua aplicação, porém não precisará de conhecer o modo como foi construído, porque a interação da interface com o modelo será estabelecida pela comunicação entre os respetivos sinais e eventos de entrada e saída.

Portanto, para além da geração dos componentes dos *widgets*, interessa garantir também a correta geração das suas entradas e saídas, que em conjunto formam o modelo da interface e, por conseguinte, o conteúdo do seu componente – veja-se a Figura 3.2. A principal vantagem será o utilizador não ter que proceder a qualquer tipo de edição do modelo ou componente da interface no IOPT-Flow Editor, a não ser a ligação dos sinais e eventos de entrada e saída a outro modelo, através do desenho de arcos.

Tome-se como exemplo, mais uma vez, as entradas e saídas associadas a um *widget* do tipo botão, como apresentado na Figura 3.12.

A variável "visible" traduz o estado de um *widget* na interface, quanto à sua aparência: se o *widget* é visível, ou não. Independentemente do valor a entrada "visible", a posição de um *widget* pode ser definida pela atribuição de valores às coordenadas x e y através das entradas com o mesmo nome; e o tamanho com que aparece será caracterizado pela dimensão definida pelos valores atribuídos as entradas "width" e "height", respetivamente, a largura e a altura do *widget*. Se estas características não forem definidas, aquando da simulação do um projeto no IOPT-Flow Editor, o ambiente calcula automaticamente o tamanho do *widget* na interface, colocando-o no canto superior esquerdo.

A variável "sensitive" permite ao componente a receção de valores por parte do utilizador, ou seja, qualquer ação que o utilizador queira realizar sobre um modelo, deverá ser realizada através da GUI; a deteção da ocorrência de uma ação será expressa pela atualização de valores nos sinais de saída do componente, tal que o seu evento de saída, se apresentar algum, notifica a alteração desso valor do sinal.

Numa GUI uma página é equivalente a uma janela ou ecrã que contém um conjunto de objetos gráficos ou *widgets* que fornecem diferentes tipos de informação e podem receber inputs do utilizador (textos, imagens, botões, caixas de seleção, entre outros). Com este componente, o que

se pretende é que num dado momento apenas uma página seja visível (a página selecionada) permitindo a criação de interfaces onde o utilizador navega por várias páginas.

Assim, torna-se possível definir o valor das entradas e saídas dos componentes no ambiente IOPT-Flow editor, ou seja, é possível informar um modelo de quais serão os valores e o tipo de entrada e de saída num componente, e, por conseguinte, na interface. Na definição das constantes e dos sinais, são, mais uma vez considerados atributos que são requeridos no ambiente IOPT-Flow Editor. Assim podem ser criados os valores de entrada "visible", "sensitive", "x", "y", "width" e "height" com os atributos "name", "mode", "type", "min", "max" e "value".

A tabela seguinte apresenta o exemplo para a atribuição de valores a cada um dos parâmetros respeitantes às entradas e saídas que caracterizam um componente do tipo botão, através da definição dos valores dos atributos de uma constante, de um sinal ou de um evento. Qualquer um dos valores editáveis pode ser alterado.

Tabela 3.1: Caracterização de cada tipo de entrada e saída para um componente IOPT-Flow do tipo botão.

Formatos	Entradas	Saídas
Nenhum	-	-
Constante Nome Tipo: Integer Range ou Boolean Mínimo, Máximo e Valor	 s002 Integer Range Min = 0, Max= 0 e Valor = 0	Formato inválido
Sinal Nome Tipo: Integer Range ou Boolean Modo: Input, Output ou Internal Mínimo, Máximo e Valor	 s003 Integer Range Input Min = 0, Max= 0 e Valor = 0	 s004 Integer Range Output Min = 0, Max= 0 e Valor = 0
Evento Nome Modo: Input, Output ou Internal I/O Pin Nr	-	 s005 Output 0

Existem 4 formatos possíveis para caracterizar uma entrada num componente IOPT-Flow, e 3 formatos para caracterizar uma saída, estando apresentados na Tabela 3.1: "Nenhum", "Constante", "Sinal", "Evento". Como se pode observar, nenhuma saída, de qualquer widget que seja configurado pode apresentar o formato de constante; na verdade, tal não é possível se pensarmos que não podemos forçar valores à saída de um componente IOPT-Flow, pois essas saídas deverão conter informação proveniente do mesmo, que caracterizam o seu estado. Por exemplo, no caso do botão, as saídas "click" e "pressed" traduzem a ocorrência de um clique e pressão no botão, respetivamente.

Na definição das entradas e saídas de um *widget*, o utilizador poderá escolher as opções que lhe são apresentadas, sendo "Nenhum" a opção por *default*. Do mesmo modo, para uma entrada do tipo sinal, apenas será possível conectar sinais de entrada ou internos, ou constantes; para uma saída do tipo sinal, poderão se conectar saídas do tipo interno ou de saída; e para entradas ou saídas do tipo evento, apenas se poderão conectar entradas e saídas do tipo evento, respetivamente.

Para além disso, como se pode observar, a uma constante não é apresentada a atribuição de um "Modo", e a um evento não se define o tipo ou limites de variação do valor.

No subcapítulo seguinte é explicado como foi implementada a geração dos *widgets*, dos sinais, dos modelos em si e do componente.

Sempre que o utilizador altera alguma propriedade ou característica, deve clicar no botão "save" para que a alteração seja registada e o parâmetro modificado seja atualizado.

Todos os parâmetros são usados para passar informação para o código XML usado como ferramenta de tradução da GUI num modelo e componente DS-Pnet.

3.4. Tradução de uma GUI para o formalismo de modelação DS-Pnet e geração do seu modelo e componente IOPT-Flow

O GUI Editor permite a edição de interfaces gráficas de utilizador com recurso a ficheiros de imagem e áudio, ao *drag-and-drop* de *widgets*, e à atribuição de valores a propriedades e entradas e saídas dos respetivos componentes DS-Pnet. A parametrização destes valores deve-se ao facto da nova ferramenta também permitir a tradução das GUI para DS-Pnet, para serem usadas no IOPT-Flow Editor.

No GUI Editor, aquando da tradução de uma GUI para DS-Pnet, nomeadamente a geração do seu modelo e componente, podem também ser gerados:

- Componentes dos *widgets*;
- Sinais de entrada, internos e saída;
- Constantes;
- Eventos de entrada e saída.

No IOPT-Flow Editor os modelos e componentes DS-Pnet são armazenados em documentos XML. Por isso, a geração de cada um dos elementos elencados anteriormente corresponde, na verdade, à criação de elementos XML.

Considere-se um exemplo simples: uma GUI que apresenta apenas um botão, como o da Figura 3.13. O botão apresenta as propriedades da Figura 3.11.

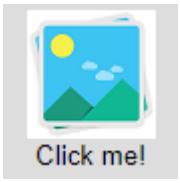


Figura 3.13: Widget do tipo botão.

A representação XML do botão – ou de qualquer *widget* – corresponde à criação de um elemento do tipo "component", como o da Figura 3.13. Um "component" apresenta sempre um conjunto de elementos filho (child) que vão definir a aparência do componente DS-Pnet no ambiente IOPT-Flow Editor: "name", "source_model", "input", "output" e "comment".

```
<component id="b001" class="ui/button.xml" x="170" y="200" width="160" height="160" rot="0"
implementation="iopt-flow" target="external" res_location="./files/gui/ui/xml_file_button/
padrao.png" param_string="@ @">
 <name off_x="-80" off_y="-85" text="ui_button"/>
 <source_model file="files/ui_button.xml"/>
 <input id="b001.Visible" name="Visible" off_x="-80" off_y="-60" type="boolean"/>
 <input id="b001.Sensitive" name="Sensitive" off_x="-80" off_y="-40" type="boolean"/>
 <input id="b001.X" name="X" off_x="-80" off_y="-20" type="range" min="0" max="1023"/>
 <input id="b001.Y" name="Y" off_x="-80" off_y="0" type="range" min="0" max="1023"/>
 <input id="b001.Width" name="Width" off_x="-80" off_y="20" type="range" min="0" max="1023"/>
 <input id="b001.Height" name="Height" off_x="-80" off_y="40" type="range" min="0"
max="1023"/>
 <input id="b001.PageNr" name="PageNr" off_x="-80" off_y="60" type="range" min="0" max="31"/>
 <output id="b001.Click" name="Click" type="event" off_x="80" off_y="-60"/>
 <output id="b001.Pressed" name="Pressed" off_x="80" off_y="-40" type="boolean"/>
 <comment text="Click me!" off_x="0" off_y="20"/>
</component>
```

Figura 3.14: Representação XML de um widget do tipo botão.

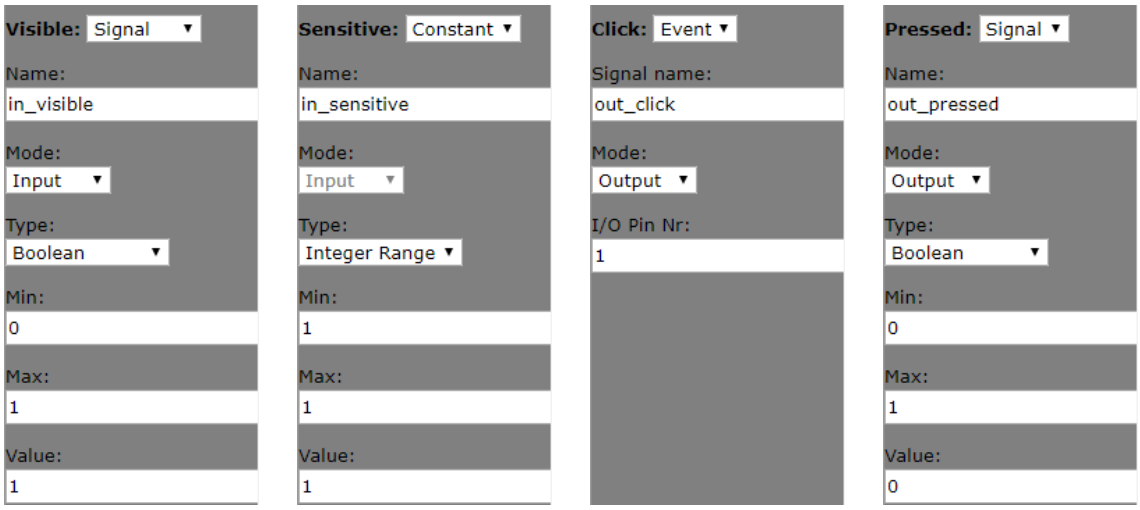
Para além disso, ao elemento "component" são associados atributos; a maioria são os mesmos que são associados ao elemento HTML que representa o widget no GUI Editor, sendo visíveis e editáveis no painel da Figura 3.11. Os valores dos atributos "rot" e "x" e "y" são definidos aquando da geração do modelo. Inicialmente, "x" é igual a 170 e "y" é igual a 200, e informam sobre a posição central do componente DS-Pnet no modelo. Conforme o modelo é gerado, esses valores são incrementados para que os componentes dos *widgets*, e os respetivos valores de entrada e saída, não apareçam sobrepostos no modelo.

A tag "name" permite que o nome do componente apareça no seu canto superior esquerdo, junto ao bordo, na posição calculada pela soma das coordenadas x e y do centro do componente com -80 e -85, respetivamente. Na verdade, todos os elementos filho de um "component" apresentam um conjunto de atributos, inclusive um par de offsets que é constante para cada elemento filho num determinado tipo de componente, e que determina a posição desse elemento filho nesse componente DS-Pnet, com exceção do "source_model". O "source_model" contém apenas a referência do modelo de origem que implementa o componente DS-Pnet. Por exemplo, neste caso, tratando-se da implementação de um componente DS-Pnet do tipo botão, o seu modelo encontra-se disponível no ficheiro "ui_button.xml" da pasta "files".

Os restantes elementos filho do "component" são o "comment" e cada um dos parâmetros da lista de entradas e saídas do componente DS-Pnet, representados pelas tags "input" e "output". Estes elementos são caracterizados por um identificador, um nome, um par offset, o tipo de valores que a entrada ou saída pode receber e, dependendo do tipo, os respetivos limites.

O exemplo continua com a atribuição feita a dois parâmetros de entrada e dois parâmetros de saída do botão no ambiente GUI Editor. Considere-se a seguinte atribuição de valores:

- A variável "visible", nomeada "in_visible", será um sinal de entrada do tipo "boolean", inicializado a 1;
- A variável "sensitive", "in_sensitive", será uma constante do tipo "integer range" com valor de 1;
- A variável "click", "out_click", será um evento de saída;
- A variável "pressed", com nome "out_pressed", será um sinal de saída do tipo "boolean", inicializado com valor 0.



(a) Entrada "visible" (b) Entrada "sensitive" (c) Saída "click" (d) Saída "pressed"

Figura 3.15: Painéis para atribuição de valores a entradas e saídas.

Assim, para além da geração do componente DS-Pnet do tipo botão, é necessário e possível criar-se elementos XML que representem cada um dos valores atribuídos aos parâmetros do painel da Figura 3.12. Estes valores, como se pode verificar, serão elementos que representam sinais, eventos e constantes no ambiente IOPT-Flow.

Para a criação das variáveis "visible", "pressed", ou de qualquer outro sinal, é usado a representação apresentada no excerto de código da Figura 3.16.

```
<signal id="in_visible" x="30" y="95" mode="input" type="boolean" min="0" max="1" value="1"
dynamic="none" frac="0"/>
<signal id="out_pressed" x="285" y="160" mode="output" type="boolean" min="0" max="1"
value="0" dynamic="none" frac="0"/>
```

Figura 3.16: Representação XML de um sinal de entrada e um sinal de saída.

Um elemento XML do tipo "signal" representa um nó *dataflow* no ambiente IOPT-Flow Editor, como apresentado na Tabela 2.1. O elemento "signal" não possui elementos filhos, mas é composto por um grupo de atributos que o caracterizam: um identificador, a sua posição gráfica no modelo gerado para IOPT-Flow Editor, o modo, tipo, valor, limites do valor, dynamic e frac.

Como se pode observar, o que caracteriza um sinal como sendo de entrada, saída ou interno é o valor do atributo "mode".

A posição gráfica dos sinais, eventos e constantes que se vão ligar a um componente no IOPT-Flow Editor, é calculada tendo em consideração o número de entradas que um componente tem, garantindo-se que a apresentação do modelo DS-Pnet da GUI no IOPT-Flow editor é ordenada e limpa.

Por sua vez, a representação XML de um evento é a que se apresenta no excerto da Figura 3.17.

```
<event id="out_click" x="310" y="140" mode="output" io_pin="1">
 <comment text="" off_x="0" off_y="20"/>
</event>
```

Figura 3.17: Representação XML de um evento.

Ao contrário de um sinal, um evento não apresenta valor nem definição de limites, porque apenas representa a ocorrência de determinada ação. Para possibilitar a validação e implementação de modelos a correrem em dispositivos remotos, o elemento "event" possui o atributo "io_pin", definido com o valor de um porto que pode detetar ou fazer ocorrer um evento.

Apresentando uma representação diferente das anteriores, o desenho de constantes num modelo DS-Pnet, segue uma organização própria de expressões matemáticas implementadas em IOPT-Flow, contendo um elemento filho "expression", neste caso, com apenas um operando que terá sempre o valor 1.

```
<operation id="o007" x="30" y="130" constant="1" rot="0" shape="rect" size="20"
locked="true">
 <name off_x="-15" off_y="-5" text="in_sensitive"/>
 <output off_x="20" off_y="0" name="K" id="o007.K" type="range" min="1" max="1"
dynamic="none" frac="0">
  <expression>
   <text>1</text>
   <operand type="literal" value="1"/>
  </expression>
 </output>
</operation>
```

Figura 3.18: Representação XML de uma constante.

Para associar cada um dos valores de entrada e saída ao componente DS-Pnet, nomeadamente estabelecer a ligação de sinais constantes e eventos a entradas e saídas do componente, é necessário implementarem-se arcos. Assim, sempre que se cria um sinal, evento ou constante é imediatamente criado um arco que liga esse elemento ao componente DS-Pnet. A Figura 3.19 apresenta os quatro arcos criados para o exemplo que está a ser analisado.

```
<arc id="a006" type="read" source="in_visible" target="b001.Visible"/>
<arc id="a008" type="read" source="o007.K" target="b001.Sensitive"/>
<arc id="a009" type="read" source="b001.Click" target="out_click"/>
<arc id="a010" type="read" source="b001.Pressed" target="out_pressed"/>
```

Figura 3.19: Representação XML de arcos.

Como se pode observar, os arcos que estabelecem a ligação destes elementos são sempre arcos de leitura ou "read". Para além de um indentificador, os arcos apenas precisam de ter informação sobre a origem do seu desenho no modelo e a que elemento se vão ligar, ou seja a "source" e o "target", senso a estes parâmetros dado o valor de identificadores de outros elementos.

Após a criação do componente IOPT-Flow do *widget* da Figura 3.13, e de cada um dos elementos definidos na Figura 3.15, inclusive as suas ligações ao componente do *widget*; ou seja, após a geração do modelo IOPT-Flow do exemplo apresentado, o respetivo ficheiro XML é guardado, e seguidamente é criado o componente IOPT-Flow do modelo.

A geração do componente IOPT-Flow do modelo da interface é similar à criação do componente IOPT-Flow do *widget* da Figura 3.13, tal que a sua representação XML segue as mesmas regras da representação apresentada na Figura 3.14 , observe-se a Figura 3.20.

```
<?xml version="1.0"?>
<?xml-stylesheet href="http://gres.uninova.pt/iopt-flow/show-pf.xsl" type="text/xsl"?><net
name="lib">
 <component id="c1" class="local/xml_file_button.xml" x="105" y="80" width="110" height="60"
rot="0" implementation="iopt-flow" target="default">
  <name off_x="-55" off_y="-35" text="xml_file_button_?"/>
  <source_model file="files/gui/xml_file_button.xml"/>
  <input id="c1.in_visible" name="in_visible" off_x="-55" off_y="-10" type="boolean"/>
  <output id="c1.out_click" name="out_click" type="event" off_x="55" off_y="-10"/>
  <output id="c1.out_pressed" name="out_pressed" off_x="55" off_y="10" type="boolean"/>
 </component>
</net>
```

Figura 3.20: Representação XML do componente IOPT-Flow da interface, gerado automaticamente.

Tal como esperado, neste exemplo, o componente IOPT-Flow gerado pelo novo ambiente, apresenta a representação da Figura 3.20, onde apenas foram definidas as entradas e saídas às quais foram indicadas a ligação com sinais ou eventos. No caso da entrada "sensitive", como lhe foi atribuído uma entrada do tipo constante, aquando da criação do componente, não é necessário apresentar se essa entrada, por não depender de nenhum valor externo.

Assim, como resultado, podemos visualizar e usar o modelo DS-Pnet do botão da Figura 3.21 no IOPT-Flow.

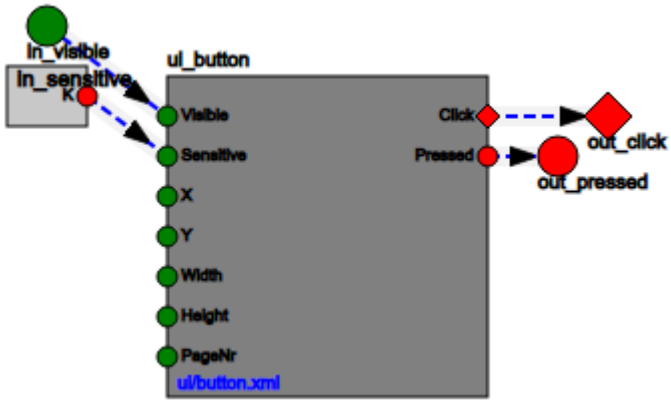


Figura 3.21: Modelo IOPT-Flow de uma interface que possui apenas um botão, com 2 parâmetros de entrada, e 2 parâmetros de saída definidos.

Do mesmo modo, a figura x, apresenta o componente gerado pelo GUI Editor.

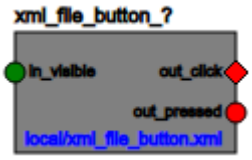


Figura 3.22: Componente IOPT-Flow, que possibilita o estabelecimento de ligações com uma entrada e duas saidas.

Aqui, não será feita a análise dos resultados obtidos neste exemplo, apenas se destaca a correta geração de cada um dos elementos e respetivas ligações.

Assim, atente-se à Figura 3.**23**, onde é apresentado um fluxograma que descreve o algoritmo implementado para a geração do modelo e componente IOPT-Flow de uma GUI. Como referido anteriormente, a conversão da GUI para DS-Pnet dá-se quando o utilizador o solicita; quando isso acontece, inicia-se a aquisição de todos os elementos, *widgets*, que se apresentam definidos na página HTML do ambiente IOPT-Flow GUI Editor; ou seja, aqueles que compõem uma GUI.

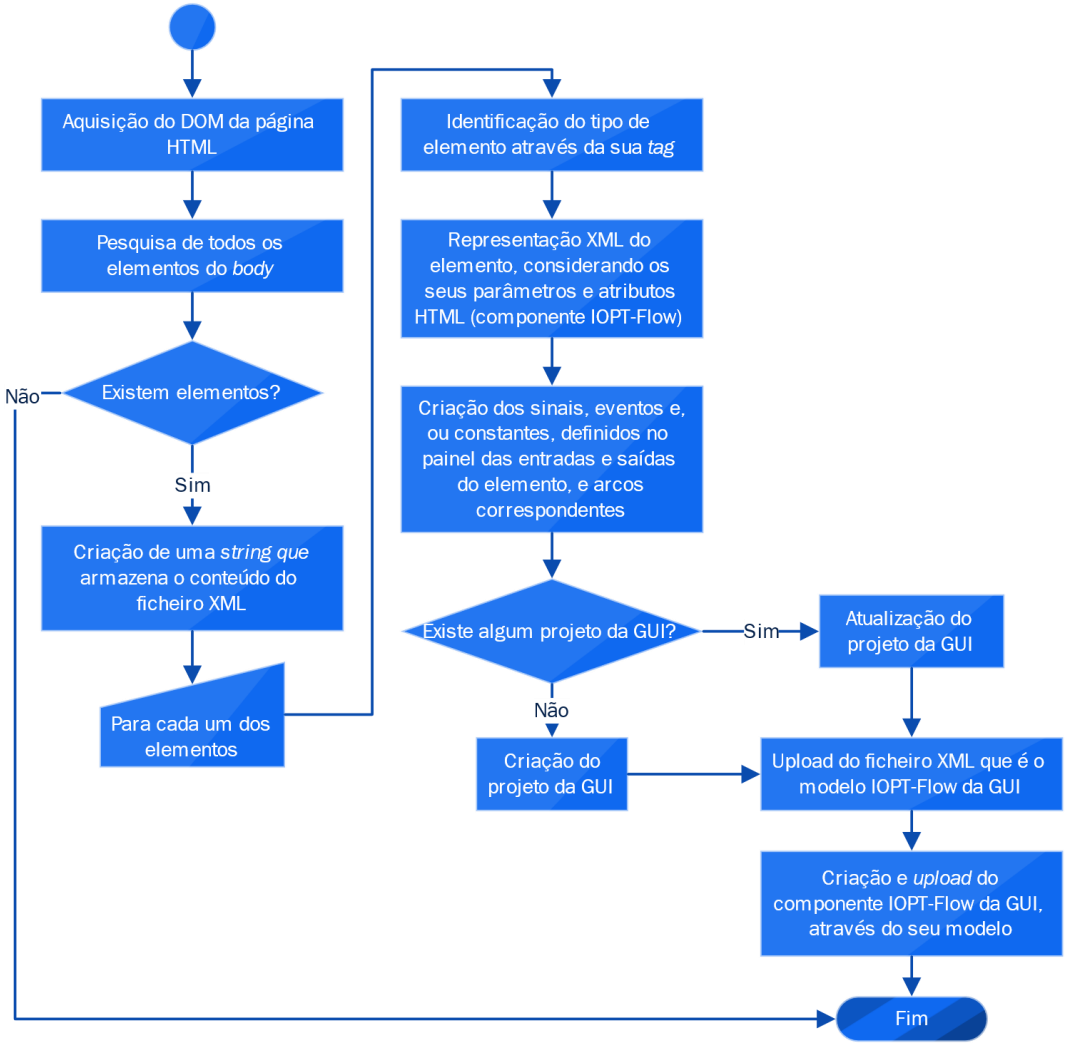


Figura 3.23: Fluxograma que apresenta o algoritmo implementado para a geração de modelos e componentes IOPT-Flow.

Após a aquisição de todos os elementos - caso existam – é identificado o tipo de *widget* que cada um representa através do seu *tag*, para possam ser representado no formato XML correspondente ao seu componente IOPT-Flow, como é apresentado na Figura 3.14, que se refere à representação XML de um *widget* do tipo botão. A sua representação – e a de todos os elementos da GUI – vai ser armazenada numa *string*, em conjunto com a representação XML de todos os *dataflow* do modelo da GUI, nomeadamente os sinais e eventos de entrada e saída, as constantes

e os respetivos arcos que estabelecem a ligação desses elementos ao componente IOPT-Flow correspondente.

Posteriormente, após a representação XML de todos os elementos da GUI, é analisado se existe algum projeto associado à mesma; se não existir, é criado um, caso contrário, o projeto é atualizado e informação contida na *string* é imediatamente usada para guardar o modelo da interface num ficheiro XML. Seguidamente, é realizada a geração do respetivo componente IOPT-Flow através da análise do conteúdo desse ficheiro. Durante o processo de armazenamento e upload do modelo e do componente, estes vão sempre substitui versões que possam já existir.

Seguidamente é apresentado o modo como são armazenados os ficheiros que fazem parte de um projeto criado no ambiente IOPT-Flow GUI Editor.

3.5. Estrutura de armazenamento dos ficheiros de um projeto

No novo ambiente, através da edição do documento da sua página HTML, pode-se criar a interface de um modelo DS-Pnet. Sempre que desejado, conforme apresentado no subcapítulo anterior, também se podem gerar os ficheiros XML que representam o modelo e o componente DS-Pnet da interface, possibilitando a sua utilização no ambiente IOPT-Flow Editor. A Figura 3.1 apresenta a relação entre o GUI Editor e o IOPT-Flow Editor, evidenciando esta perspetiva.

Apesar dos ficheiros XML serem criados a partir da informação contida no ficheiro HTML, um projeto de uma GUI no novo ambiente pode ser continuamente editado sem afetar o estado atual do respetivo modelo e componente DS-Pnet, que só são gerados ou atualizados quando o utilizador o indicar. O oposto também se verifica, visto que existindo um modelo do projeto em DS-Pnet, é possível editá-lo no ambiente IOPT-Flow Editor sem modificar o desenho da GUI no novo ambiente.

Assim, os ficheiros HTML e XML não são alterados pelas ações que são efetuadas nos ambientes IOPT-Flow Editor e GUI Editor, respetivamente. Para gerir o acesso dos dois ambientes aos ficheiros de desenvolvimento de uma GUI, nomeadamente os ficheiros HTML no GUI Editor e os ficheiros XML no IOPT-Flow Editor, foram criadas duas pastas – "ui" e "gui" – adicionadas à pasta "files", já existente: a pasta "ui" dentro da pasta "gui", dentro da pasta "files". A Figura 3.24 apresenta um esquema representativo da organização do armazenamento dos ficheiros de um projeto, inclusive os recursos usados e acedidos pelos dois ambientes.

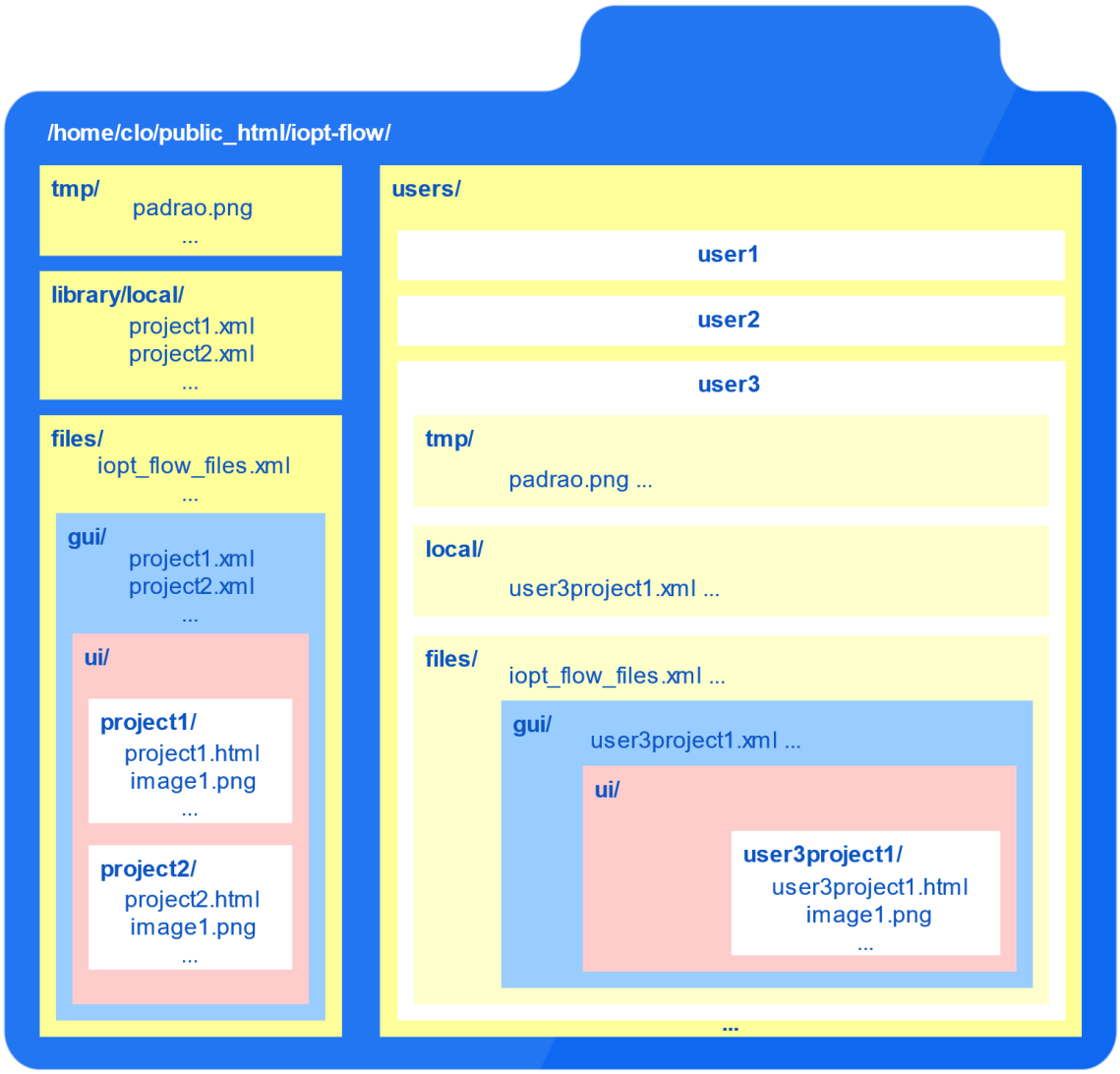


Figura 3.24: Esquema da organização das pastas e ficheiros de projetos.

O esquema da Figura 3.24 foi desenhado considerando dois cenários, tal que o utilizador do GUI Editor e IOPT-Flow Editor pode estar a usar os ambientes com sessão iniciada, ou não. Atente-se ao seguinte:

- A pasta "files" contém os modelos desenvolvidos no IOPT-Flow Editor e a pasta "gui";
- A pasta "gui" armazena os modelos DS-Pnet gerados no novo ambiente e a pasta "ui", com os projetos desses modelos;
- Na pasta "ui" encontram-se as pastas dos projetos desenvolvidos no GUI Editor – uma pasta para cada projeto;
- Na pasta de cada projeto encontra-se o ficheiro HTML usado na edição da GUI e os respetivos recursos;
- Na pasta "local" encontra-se o componente da interface gerado no GUI Editor;
- Na pasta "tmp" encontram-se os recursos que podem ser usados na elaboração de uma GUI, caso não exista nenhum projeto associado à mesma;
- Em qualquer caso, os recursos podem ser adicionados por upload;

- Caso o utilizador tenha sessão iniciada, as pastas “files”, “tmp” e “local” encontram-se dentro da pasta do “user”;
- O nome que o utilizador dá a um projeto é usado para nomear o componente e o modelo DS-Pnet, facilitando a sua procura e atualização.

4

Testes e Exemplos de Validação

O presente capítulo apresenta a fase de testes e validação da implementação do editor e gerador automático de interfaces gráficas de utilizador. Para validar o ambiente proposto e desenvolvido neste trabalho, foram realizados vários testes com exemplos de complexidades diferentes que apoiaram as várias etapas na implementação do ambiente. Neste capítulo apresentam-se dois dos exemplos, e as suas aplicações. Para ambos os casos, foi usada a ferramenta desenvolvida e o ambiente IOPT-Flow Editor, que permite visualizar os resultados obtidos, nomeadamente os modelos e componentes DS-Pnet gerados automaticamente pelo ambiente IOPT-Flow GUI Editor; para além disso, também oferece ferramentas que apoiam a simulação de modelos, e através do gerador de código C automático possibilita a validação remota dos mesmos.

O primeiro exemplo de aplicação apresentado diz respeito à criação de uma interface gráfica de utilizador usada para o controlo de um jogo pong. Como o jogo foi implementado no âmbito de outra dissertação, inclusive o seu modelo e a sua interface foram desenhadas no ambiente IOPT-Flow Editor, torna-se um excelente exemplo de aplicação, podendo-se comparar a interface pré-existente, desenhada manualmente, com a criada no novo ambiente, gerada automaticamente.

Posteriormente, foi desenvolvida uma GUI que pode ser aplicada à visualização e monitorização de parâmetros associados ao funcionamento de uma cadeira de rodas elétrica. Para a implementação deste exemplo, também foi desenvolvido um modelo DS-Pnet que permite modelar o comportamento da interface. No final, para além da sua simulação no ambiente IOPT-Flow Editor, o exemplo foi executado através da sua implementação num dispositivo Raspberry Pi 3, e através do ambiente IOPT-Flow Editor foi possível observar o seu comportamento remotamente.

As interfaces apresentadas neste capítulo estão disponíveis a consulta em http://gres.uninova.pt/~clo/iopt-flow/inter_clo.php. Do mesmo modo, todos os modelos e componentes DS-Pnet estão disponíveis em http://gres.uninova.pt/~clo/iopt-flow/; estando os componentes localizados na pasta "local" da biblioteca.

4.1. Interface gráfica de utilizador do jogo pong

Como primeiro exemplo de aplicação para validação do trabalho desenvolvido, foi implementada a interface gráfica de utilizador de um jogo pong. No âmbito do trabalho desenvolvido numa dissertação de doutoramento referenciada nesta tese, "The DS-Pnet modelling formalism for cyber-phisical system development", foi apresentada como aplicação de validação um jogo pong *multi-player*.

Aquando da sua implementação, o jogo foi criado usando as ferramentas disponibilizadas pelo IOPT-Flow Editor, incluindo os componentes de GUI DS-Pnet da biblioteca e o gerador de código C automático, entre outros. Seguidamente é apresentado esse jogo.

4.1.1. Jogo Pong desenvolvido e simulado no ambiente IOPT-Flow Editor

No início da criação do jogo pong *multi-player*, este foi implementado considerando a sua jogabilidade para apenas um jogador, tendo sido validada a sua implementação com sucesso. Sabendo que as interfaces usadas para os dois jogadores são iguais e implementadas com o mesmo componente DS-Pnet, para validar o trabalho desenvolvido nesta tese, considerou-se apenas a implementação do jogo para um jogador.

A primeira versão do modelo do jogo pong para um jogador, antes de se separar a parte que modela o funcionamento do jogo, da parte que compõem a interface de utilizador, é a que se apresenta na Figura 4.1, e está disponível abrindo o ficheiro "pong.xml".

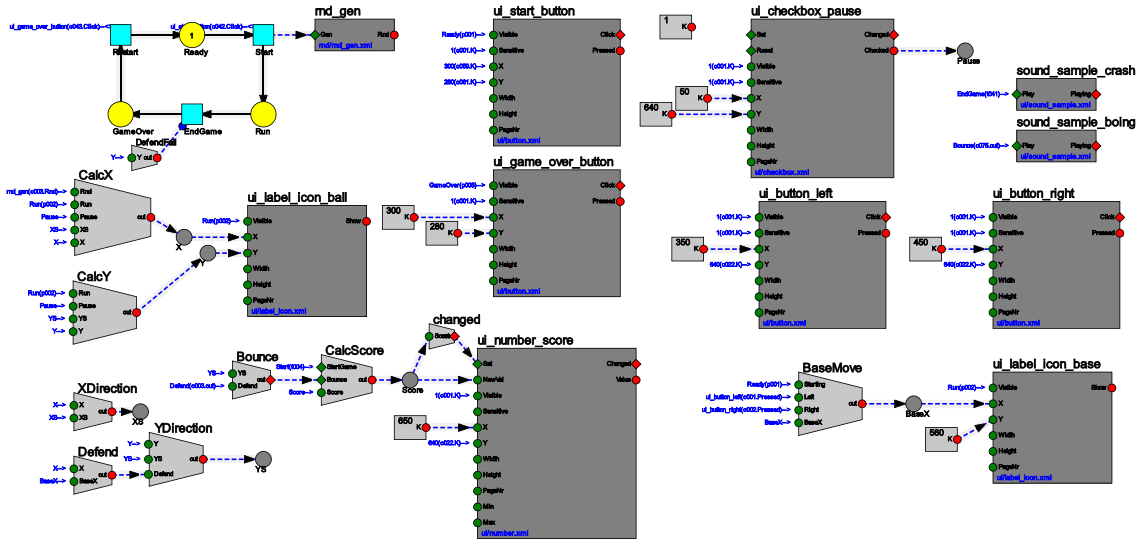


Figura 4.1: Modelo DS-Pnet do jogo pong para um jogador com interface incluída, desenhado no IOPT-Flow Editor.

A maioria dos elementos que compõem o modelo da Figura 4.1 são componentes retangulares cinzentos escuros, que correspondem à interface gráfica de utilizador que apresenta a janela do jogo. A interface é composta por uma bola, uma base, a pontuação do jogador, e os botões de pausa, deslocamento da base para a esquerda e para a direita, fim do jogo e início do jogo. A bola

e a base são componentes DS-Pnet do tipo imagem; a pontuação é dada por um componente do tipo número; o botão de pausa, corresponde, na verdade, a um componente do tipo *checkbox*; e os restantes elementos correspondem a componentes do tipo botão. No canto superior direito do modelo também se podem observar dois componentes DS-Pnet do tipo áudio, usados para dinamizar o jogo e caracterizar o embate da bola nas paredes e na base, ou quando se perde o jogo.

O restante modelo é composto por uma rede de Petri, operações, constantes e sinais internos, que apoiam a modelação do funcionamento do jogo. Os sinais internos e a ausência de sinais e eventos de entrada ou de saída ajudam a perceber que o modelo da Figura 4.1 não poderá ser afetado por sinais ou eventos externos, mesmo que provenientes de outros modelos que possam também ser desenhados no ambiente IOPT-Flow Editor. Efetivamente, observando a Figura 4.2, podemos observar o componente do modelo anterior, com particular atenção ao facto de este não apresentar quaisquer sinais nas extremidades do seu desenho.

Figura 4.2: Componente DS-Pnet do jogo pong para um jogador com interface incluída.

Ora, sendo um dos objetivos do formalismo DS-Pnet o uso de componentes para simplificar implementações, permitindo o uso dos mesmos como submodelos; e o estabelecimento de ligações entre modelos implementados em diferentes alturas ou por diferentes formalismos e linguagens de desenvolvimento, considerar-se-á de seguida o mesmo jogo, separado em dois componentes: um componente que modela o funcionamento do jogo, e outro que apresenta a sua interface e possibilita a interação com um utilizador.

De seguida, a Figura 4.3 apresenta o modelo DS-Pnet da interface do jogo pong que foi retirada do modelo da Figura 4.1, através da sua separação do restante modelo.

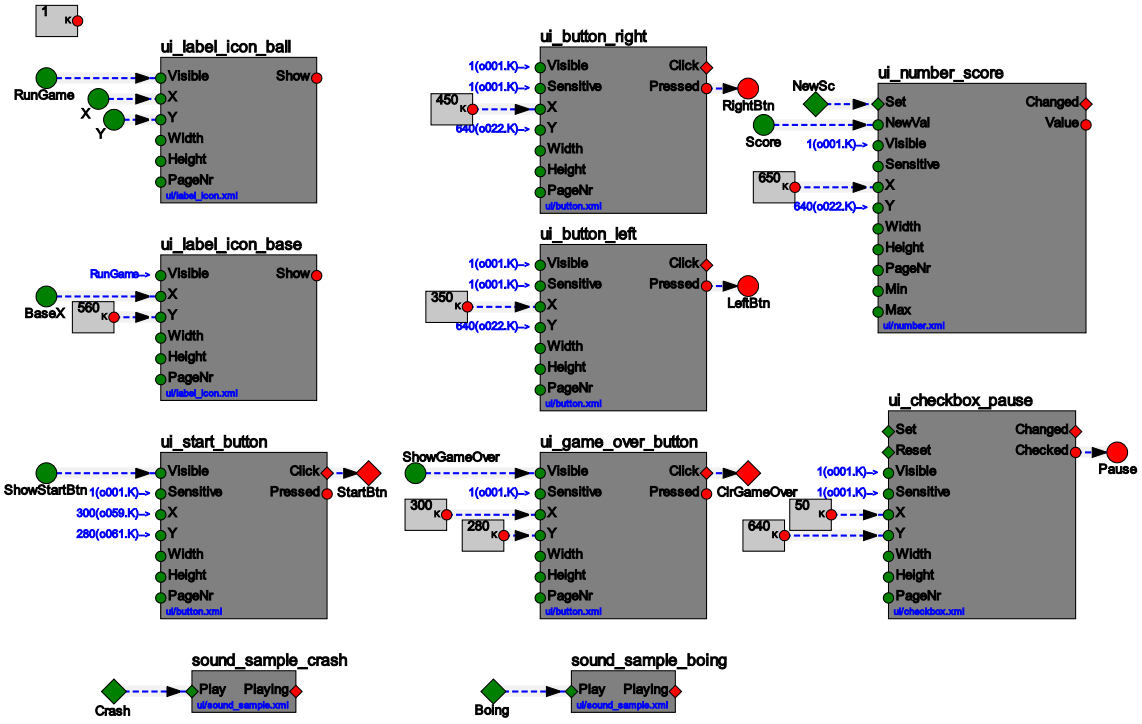


Figura 4.3: Modelo DS-Pnet da interface do jogo pong, desenvolvido no IOPT-Flow Editor.

O modelo anterior está disponível no ambiente IOPT-Flow Editor ao abrir o ficheiro "pong_if.xml"; o seu componente é apresentado na Figura 4.4.

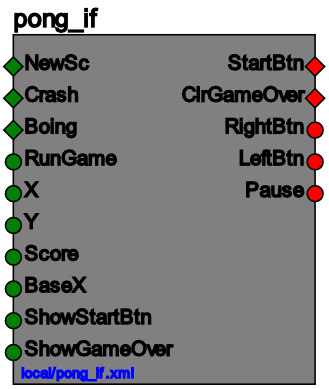


Figura 4.4: Componente DS-Pnet do modelo da interface do jogo pong, criado no IOPT-Flow Editor.

Neste caso, já se observam sinais e eventos de entrada e saída, nomeadamente os que afetam a aparência da interface, e os que possibilitam a alteração do estado do modelo funcional do jogo, respetivamente

O evento "NewSc" deteta a nova pontuação do jogador, dada pelo sinal "Score", e atualiza-la na interface; os eventos "Crash" e "Boing" iniciam a reprodução dos sons do jogo; o sinal "RunGame" permite visualizar a base e a bola durante o jogo, cuja posição é dada pelos sinais "X", "Y"; por sua vez, o sinal "BaseX" permite modificar a posição horizontal da base; e os sinais "ShowStartBtn" e "ShowGameOver" permitem visualizar, ou não, os botões de início e fim do jogo, respetivamente.

Do mesmo modo, os eventos de saída "StartBtn" e "ClrGameOver" ocorrem quando é detetado o clique nos botões de começo e fim de jogo, respetivamente; o sinal "Pause", informa sobre o estado da *checkbox* usada para parar ou retomar um jogo; e os sinais "LeftBtn" e "RightBtn" informam sobre a pressão efetuada nos botões que deslocam a base para a esquerda e para a direita, respetivamente.

Para validar o trabalho desenvolvido, não necessitamos de saber como está organizado o modelo correspondente ao funcionamento do jogo, apenas o significado dos seus sinais e eventos de entrada e saída. Por isso, aqui não se apresenta o modelo da parte "engine" do jogo, que, por outro lado, se pode observar como parte integrante do modelo da Figura 4.1. A Figura 4.5 apresenta o seu componente DS-Pnet, nomeado "pong_engine.xml".

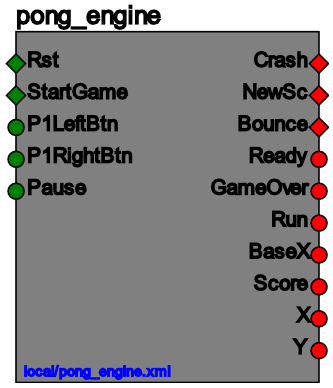


Figura 4.5: Componente DS-Pnet do modelo funcional do jogo pong, criado no IOPT-Flow Editor.

O componente da Figura 4.5 apresenta um conjunto de entradas e de saídas que fazem correspondência às saídas e entradas do componente da Figura 4.4, respetivamente. Assim, a implementação do jogo numa perspetiva de alto nível, onde apenas se juntam os dois componentes, estabelecendo ligações entre as suas entradas e saídas, é apresentada na Figura 4.6, cujo modelo está disponível abrindo o ficheiro "pong_m.xml".

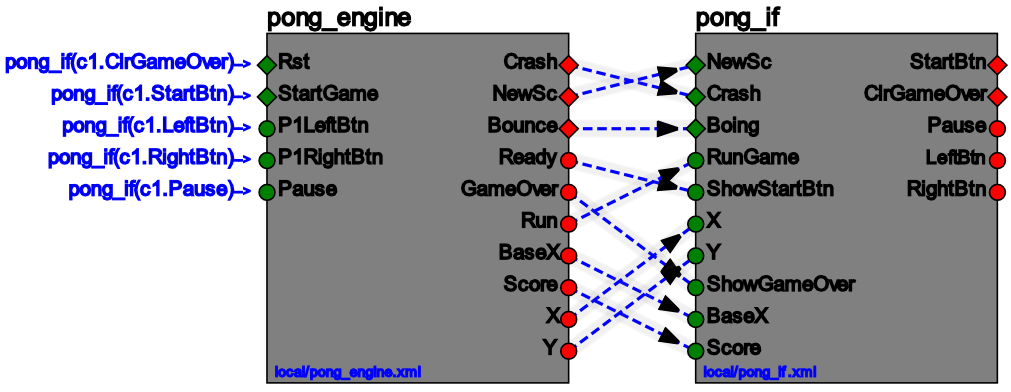


Figura 4.6: Modelo DS-Pnet do jogo pong para um jogador, composto por dois componentes DS-Pnet, desenhado no IOPT-Flow Editor.

O resultado da implementação do jogo é o da Figura 4.7.

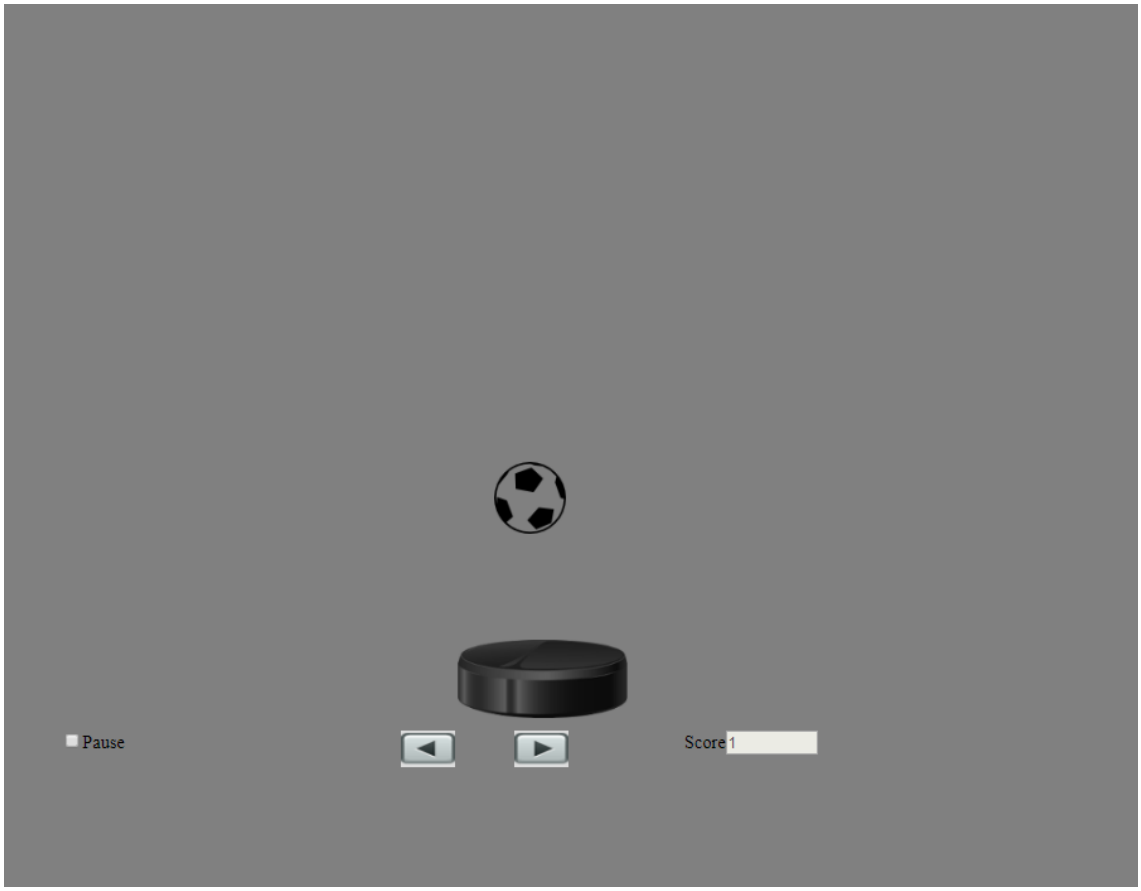


Figura 4.7: Resultado da simulação do jogo pong no ambiente IOPT-Flow Editor.

Simulando o modelo anterior, confirmou-se a boa jogabilidade do jogo desenvolvido no ambiente IOPT-Flow editor. De seguida, apresenta-se a implementação da interface do mesmo jogo, usando as ferramentas implementadas nesta tese, disponibilizadas pelo GUI Editor. Será usado o mesmo componente que modela o jogo – a parte "engine" – substituindo-se apenas a interface que existia, por uma nova, gerada automaticamente no novo ambiente.

4.1.2. Jogo pong com interface desenvolvida no ambiente IOPT-Flow GUI Editor

Para implementar a nova interface do jogo – uma interface gerada automaticamente pelo ambiente IOPT-Flow GUI Editor – foi considerado o mesmo número de componentes DS-Pnet do tipo GUI que foi usado no modelo apresentado na subsecção anterior.

Para que se perceba como foi criada a interface no novo ambiente, observe-se a Tabela 4.1. A Tabela 4.1 apresenta a parametrização de todos os *widgets*, nomeadamente a atribuição de valores aos parâmetros do painel de apresentação e configuração das propriedades e características dos *widgets*.

Tabela 4.1: Parametrização dos *widgets* da interface para o jogo pong.

Widget	**Componente DS-Pnet**	**Entradas**		**Saídas**	**Parâmetros**
Bola	Imagem	Visible	Sinal: "RunGame" Valor: 0 entre [0;1]		Nome: ui_label_icon_ball Recurso: ball.png Comentário: Ball
		X e Y	Sinal: "X" ou "Y" Valor: 0 entre [-1;801]		
Base	Imagem	Visible	Sinal "RunGame2" Valor: 0 entre [0;1]		Nome: ui_label_icon_base Recurso: base.png
		X	Sinal: "BaseX" Valor: 400 entre [-1;800]		
		Y	Constante de valor 560		
Botão de começo	Botão	Sensitive	Constante de valor 1	Click Evento: "StartBtn" ou "ClrGameOver"	Nome: ui_start_button Recurso: ball.png Param String: @ @ Comentário: Start Game!
		X	Constante de valor 300		
		Y	Constante de valor 280		
		Visible	Sinal: "ShowStartBtn" ou "ShowGameOver" Valor: 0 entre [0;1]		
Botão de fim		Sensitive	Constante de valor 1		Nome: ui_game_over_button Recurso: end.jpg Param String: @C[@ Comentário: GAME OVER
		X	Constante de valor 300		
		Y	Constante de valor 280		
Botão para a esquerda	Botão	X	Constante de valor 350	Pressed Sinal: "LeftBtn" ou "RightBtn" Valor: 0 entre [0;1]	Nome: ui_button_left Recurso: left.png Param String: @a@
		Y	Constante de valor 640		
Botão para a direita		Visible e Sensitive	Constante de valor 1		Nome: ui_button_right Recurso: right.png Param String: @s@
		X	Constante de valor 450		
		Y	Constante de valor 640		
Pontuação	Número	Set	Evento: "NewSc"		Nome: ui_number_score Comentário: Score
		NewVal	Sinal: "Score" Valor: 0 entre [0;10000]		
		Visible	Constante de valor 1		
		X	Constante de valor 650		
		Y	Constante de valor 640		
Pausa	Checkbox	Visible e Sensitive	Constante de valor 1	Checked Sinal: "Pause" Valor: 0 entre [0;1]	Nome: ui_checkbox_pause Comentário: Pause
		X	Constante de valor 50		
		Y	Constante de valor 640		
Som do jogo	Áudio	Play	Evento: "Crash" ou "Boing"		Nome: sound_sample_crash Recurso: crash.wav
Som final					Nome: sound_sample_boing Recurso: boing.wav

Considerado os valores tabelados acima, entrado no novo ambiente, ao invés de se criar qualquer projeto, iniciou-se imediatamente a edição da GUI da Figura 4.8.

Figura 4.8: Interface gráfica de utilizador do jogo pong, desenvolvida no IOPT-Flow GUI Editor.

Primeiro foram adicionados os *widgets* à área de desenho, tendo sido arrastados para a zona onde deveriam de aparecer; de seguida, foi feito o *upload* de todos os recursos necessários para dinamizar a interface; e posteriormente foram definidos os parâmetros dos *widgets* e das entradas e saídas dos respetivos componentes DS-Pnet. Seguidamente, apresentam-se os resultados obtidos.

4.1.2.1. Resultados obtidos e simulação no ambiente IOPT-Flow Editor

Após criado o desenho da interface desejada, clicou-se no botão que promove a conversão da mesma para um modelo e componente DS-Pnet. Como a interface não estava guardada antes de ser editada, aquando da sua conversão para DS-Pnet, foi criado primeiro o seu projeto, nomeado "pong_if_auto"; e só depois foram gerados o seu modelo e componente.

O projeto foi guardado na pasta "ui", juntamente com os seus recursos, alocados da pasta "tmp"; e o seu modelo e componente foram guardados com o mesmo nome na pasta "gui" e na pasta "local" da biblioteca, respetivamente. Esta gestão de ficheiros é apresentada no capítulo anterior, na Figura 3.24. Ao abrir o ficheiro "pong_if_auto.xml", é possível visualizar o modelo da Figura 4.9.

Figura 4.9: Modelo DS-Pnet da interface do jogo pong, gerado automaticamente pelo ambiente IOPT-Flow GUI Editor.

Como se pode observar, na Figura 4.9, o modelo gerado é composto por 10 componentes, tal como seria previsto; todos os componentes e elementos DS-Pnet posicionam se à mesma distância uns dos outros, automaticamente calculada para que o modelo fosse visível na área de desenho do ambiente IOPT-Flow Editor.

Do mesmo modo, através da Tabela 4.1 e do modelo da Figura 4.3, desenhado previamente no ambiente IOPT-Flow editor, podemos validar a correta geração da interface desenhada no

novo ambiente, verificando que a cada *widget* corresponde o componente DS-Pnet esperado; e que os sinais e eventos de entrada e saída do modelo apresentam os valores e parâmetros definidos no novo ambiente.

Para além disso, confirma-se uma vantagem do uso da nova interface, que assenta no facto do modelo DS-Pnet ter sido criado automaticamente não havendo erros de ligação entre os componentes e os elementos de entrada e saída dos mesmos.

O componente do modelo da Figura 4.9, também gerado automaticamente, apresenta-se na Figura 4.10.

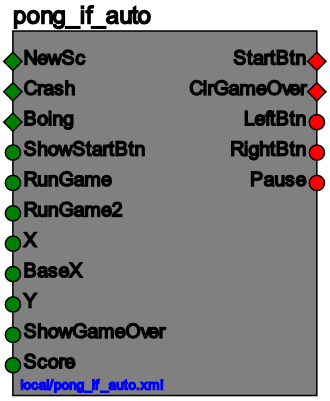


Figura 4.10: Componente DS-Pnet do modelo da interface do jogo pong, gerado automaticamente pelo ambiente IOPT-Flow GUI Editor.

Por outro lado, o componente gerado contém mais uma entrada do que o componente da interface usada anteriormente: "RunGame2", usado para tornar a base visível durante o jogo, anteriormente garantido pelo sinal "RunGame". Isto deve-se ao facto de no IOPT-Flow Editor ser possível ligar-se sinais e eventos de entrada, ou constantes, a múltiplas entradas dos componentes no modelo. No entanto, na nova ferramenta isso não é possível. Quando é atribuída uma entrada a um parâmetro de um *widget*, essa entrada é única para esse parâmetro, sendo impossível partilhá-la com outras entradas de componentes que compõem o modelo, mesmo que o seu valor e configurações sejam os mesmos que os que são usados por outro sinal, evento ou constante.

Confirmando-se a correta geração da interface do jogo, nomeadamente a correta representação XML do seu modelo e componente DS-Pnet, realizou-se a simulação do modelo do jogo com a nova interface, esquematizado na Figura 4.11, disponível no ficheiro "pong_auto.xml". A parte de modelação do funcionamento do jogo manteve-se a mesma através do uso do componente "pong_engine.xml".

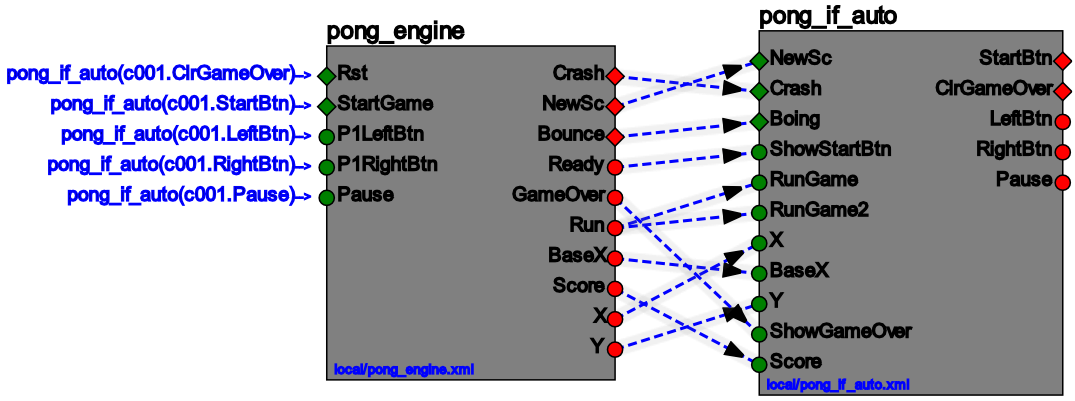


Figura 4.11: Modelo DS-Pnet do jogo pong para um jogador, com componente da interface gerado automaticamente pelo ambiente IOPT-Flow GUI Editor.

Ao simular o jogo, obtiveram-se exatamente os mesmos resultados que o modelo com a interface desenvolvida no ambiente IOPT-Flow Editor, sendo que foi apenas necessário cerca de 13 minutos para desenvolver a interface, considerando exatamente as mesma parametrizações e configurações da Tabela 4.1; algo que de certa forma morou o desenvolvimento.

A Figura 4.12 apresenta alguns dos resultados obtidos na simulação do jogo.

(a) Página inicial (b) Página durante o jogo (c) Página quando termina o jogo

Figura 4.12: Resultado da simulação do jogo pong no ambiente IOPT-Flow Editor.

Seguidamente é apresentada a segunda aplicação de validação do trabalho desenvolvido, que corresponde à criação de uma interface gráfica de utilizador para cadeiras de rodas elétricas.

4.2.Interface gráfica de utilizador para cadeiras de rodas elétricas

Neste subcapítulo é apresentado o segundo exemplo de aplicação para a validação do trabalho desenvolvido nesta dissertação. O exemplo consiste no desenvolvimento de uma interface gráfica de utilizador para cadeiras de rodas elétricas que permite alterar o modo e perfil de funcionamento de uma cadeira, saber o estado do carregamento da bateria, a sua velocidade, e observar gráficos correspondentes a medições. Assim, a Figura 4.13 apresenta, atempadamente, o desenho da interface criada.

(a) Página inicial (b) Perfil 2 ativado

(c) Modo 1 (d) Modo 2 (e) Página para apresentar gráficos

Figura 4.13: Páginas da interface gráfica de utilizador para cadeiras de rodas elétricas.

O objetivo do funcionamento da interface da Figura 4.13 é o seguinte:

- Na página inicial – (a) – ao centro é apresentado o valor da velocidade da cadeira, e são disponibilizados três botões;
- O botão "Mode" permite alterar o modo de funcionamento da cadeira entre o modo inicial, o modo 1 e o modo 2;
- O botão "Profile" altera o perfil de funcionamento entre o perfil inicial, e o perfil que permite aumentar a gama de velocidade da cadeira – (b);
- O botão com o *icon* de gráficos permite aceder à página usada para apresentar gráficos sobre medições;

- O modo de funcionamento 1 – (c) – permite que seja efetuada a elevação do assento da cadeira;
- O modo de funcionamento 2 – (d) – permite que seja efetuada a inclinação do encosto da cadeira;
- Na página usada para apresentar gráficos – (d) – existe um par de *checkboxes* que permitem selecionar o gráfico ou gráficos que se desejam observar, respeitantes à temperatura do motor e da bateria da cadeira, e um botão que possibilita o retrocesso para a página inicial;
- A qualquer altura é apresentado no canto superior esquerdo o estado do carregamento da cadeira.

Seguidamente apresenta-se o desenvolvimento da interface no novo ambiente.

4.2.1. Desenho e geração da interface desenvolvida no ambiente IOPT-Flow GUI Editor

Considerando-se as especificações apresentadas, acedendo ao editor criado neste trabalho, desenhou-se a interface da Figura 4.13, que no novo ambiente aparece como na Figura 4.14.

Figura 4.14: Interface gráfica de utilizador para cadeiras de rodas elétricas, desenhada no ambiente IOPT-Flow GUI Editor.

Antes de se proceder à explicação sobre o desenho e criação da interface da Figura 4.13, saiba-se desde já, que o componente DS-Pnet da interface, gerado no novo ambiente, foi posteriormente integrado com outro componente, cujo modelo foi desenhado no ambiente IOPT-Flow Editor. À semelhança do que acontece com o jogo pong, esse modelo deverá garantir o correto funcionamento da interface, focando o objetivo para o qual foi criada.

Para que fosse possível integrar esse modelo com a interface desenvolvida no ambiente GUI Editor, cada *widget* da interface foi configurado tal que, foram definidos os valores atribuídos aos

parâmetros e entradas e saídas dos respetivos componentes DS-Pnet no modelo. Essa parametrização e configuração está expressa na Tabela 4.2.

Tabela 4.2: Parametrização dos *widgets* da interface para cadeiras de rodas elétricas.

Widget	**Componente DS-Pnet**	**Entradas**		**Saídas**	**Parâmetros**
Botões "Mode" e "Profile"	Botão	Visible	Sinal: "VisibleModeBtn" ou "VisibleProfile" Valor: 0 entre [0;1]	Click Evento: "ClickMode" ou "ClickProfile"	Nome: ui_button_mode e ui_button_profile Recurso: cadeira.png e profile.png Param String: @ @ Comentário: Mode e Profile
		Sensitive	Constante de valor 1		
		X	Constante de valor 67 e 160		
		Y	Constante de valor 200		
Botão "Graphs" e botão de retorno	Botão	Visible	Sinal "VisibleGraphBtn" e "VisibleBackBtn" Valor: 0 entre [0;1]	Click Evento: "ClickGraphs" e "ClickBack"	Nome: ui_button_graphs e ui_button_back Recurso: graphs.png e back.png Param String: @ @
		Sensitive	Constante de valor 1		
		Y	Constante de valor 160 e 30		
Modo 1 e Modo 2	Botão	Visible	Sinal: "VisibleMode1Icon" ou "VisibleMode2Icon" Valor: 0 entre [0;1]		Nome: ui_button_mode1 e ui_button_mode2 Recurso: mode1.png e mode2.png Comentário: "Move backrest" e "Raise seat"
		X	Constante de valor 53		
		Y	Constante de valor 70		
Perfil 2	Imagem	Visible	Sinal: "VisibleFasterIcon" Valor: 0 entre [0;1]		Nome: ui_label_icon_faster Recurso: faster.png
		X	Constante de valor 113		
		Y	Constante de valor 160		
Bateria	Imagem	Visible	Constante de valor 1		Nome: ui_label_icon_battery Recurso: battery.png
Fundo	Imagem	Visible	Sinal: "VisibleBackgroundIcon" Valor: 0 entre [0;1]		Nome: ui_label_icon_backgroung Recurso: background.png
		X	Constante de valor 70		
		Y	Constante de valor 30		
Valor da bateria	Número	Set	Evento: "SetBattery"		Nome: ui_number_battery
		NewVal	Sinal: "NewValBattery" Valor: 0 entre [0;100]		
		Visible	Constante de valor 1		
		X	Constante de valor 25		
		Y	Constante de valor 6		
		Min	Constante de valor 0		
		Max	Constante de valor 100		
Valor da velocidade	Número	Set	Evento: "SetVelocity"		Nome: ui_number_velocity
		NewVal	Sinal: "NewValVelocity" Valor: 0 entre [0;100]		
		Visible	Sinal: "VisibleVelocityNumber" Valor: 0 entre [0;1]		
		X	Constante de valor 92		
		Y	Constante de valor 70		
Gráfico	Gráfico	NewSample	Evento: "NewSample"		Nome: ui_scope_graphs
		ChA e ChB	Sinal: "MotorTemperature" e "MotorBattery"		

			Valor: 0 entre [0;1000]		
		NSamples	Constante de valor 200		
		Visible	Sinal: "VisibleGraphScope" Valor: 0 entre [0;1]		
		X	Constante de valor 13		
		Y	Constante de valor 67		
Ver temperatura do motor e da bateria	Checkbox	Set	Evento: "SetMotorT" e "SetBatteryT"	Checked Sinal: "CheckedMotorT" e "CheckedBatteryT" Valor: 0 entre [0;1]	
		Reset	Evento: "ResetMotorT" e "ResetBatteryT"		
		Visible	Sinal: "VisibleMotorTCheckbox" e "VisibleBatteryTCheckbox" Valor: 0 entre [0;1]		
		Sensitive	Constante de valor 1		
		X	Constante de valor 13 e 146		
		Y	Constante de valor 250		

Assim, e como se pode observar na Figura 4.13, a interface é composta por 14 *widgets*, sendo portanto expectável a geração de 14 componentes DS-Pnet no desenho do modelo. Para além disso, os componentes cujas entradas e saídas foram definidas na Tabela 4.2, apresentarão ligação a sinais, eventos ou constantes através de arcos. Por conseguinte, o componente DS-Pnet do modelo deverá apresentar 23 entradas e 6 saídas.

4.2.2. Resultados obtidos

Após criado o desenho da interface e de serem parametrizados todos os seus elementos, no menu do ambiente IOPT-Flow GUI Editor clicou-se no botão que permite a conversão da interface para um modelo e componente DS-Pnet. À semelhança do que aconteceu no exemplo 1, primeiro foi criado o projeto da interface, nomeado "chair_if_auto"; e só depois foram gerados o seu modelo e componente.

O projeto foi guardado na pasta "ui", juntamente com os seus recursos, alocados da pasta "tmp"; e o seu modelo e componente foram guardados com o mesmo nome na pasta "gui" e na pasta "local" da biblioteca, respetivamente. Na Figura 4.15 é apresentado o modelo da interface para cadeiras de rodas elétricas; e na Figura 4.16 o seu componente.

Figura 4.15: Modelo DS-Pnet da interface para cadeiras de rodas gerado automaticamente pelo ambiente IOPT-Flow GUI Editor.

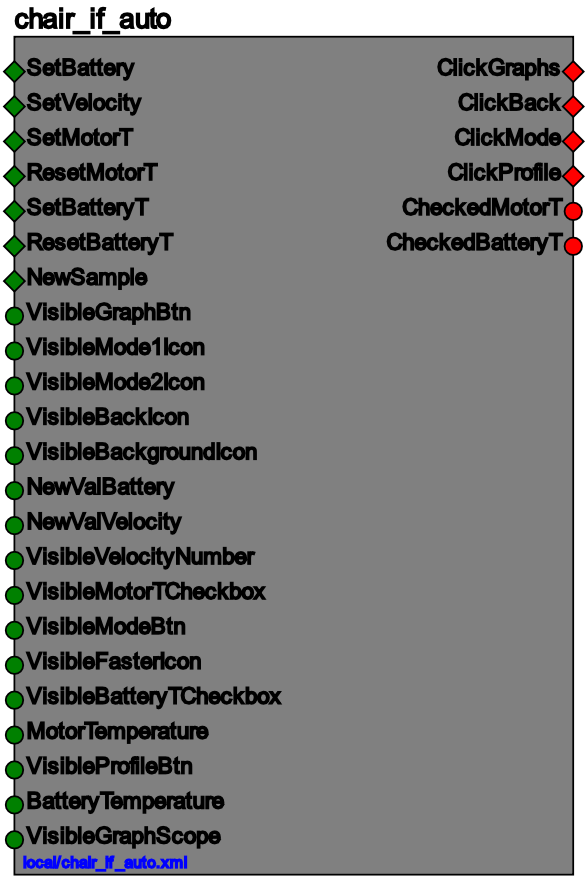


Figura 4.16: Componente DS-Pnet da interface gráfica de utilizador para cadeiras de rodas elétricas, gerado automaticamente pelo ambiente IOPT-Flow GUI Editor.

Como mencionado, o componente da interface, gerado no GUI Editor, possui 23 entradas e 6 saídas. Das 23 entradas, 7 são eventos usados para atualizar os valores apresentados na interface, nomeadamente o valor do carregamento da bateria, da velocidade da cadeira, e os valores que formam os gráficos apresentados no scope; a atualização é gerida através dos eventos "SetBattery", "SetVelocity", "SetMotorT", "SetBatteryT" e "NewSample", em simultâneo com os eventos "ResetMotorT" e "ResetBatteryT", usados para os gráficos deixarem de ser apresentados.

Todos os sinais de entrada cujo o nome se inicia com "Visible" servem para coordenar o aparecimento de um determinado elemento na interface, cujo nome também é referenciado no nome do sinal. Os restantes sinais de entrada "NewValBattery", "NewValVelocity", "MotorTemperature" e "BatteryTemperature" correspondem aos valores apresentados na interface aquando da ocorrência dos eventos indicamos anteriormente.

Do mesmo modo, o componente da Figura 4.16 apresenta 4 eventos de saída que detetam um clique num botão da interface e 2 sinais de saída que permitem perceber quais os gráficos que terão de ser apresentados no scope, ou seja, quais as *checkboxes* selecionadas pelo utilizador na página (e) da Figura 4.13.

Todos os sinais apresentados no componente estão disponíveis para estabelecer ligação com qualquer outro modelo, componente ou elemento DS-Pnet. Para esta aplicação de validação foi

necessário desenhar-se um modelo no ambiente IOPT-Flow Editor que interagisse com a interface, e modelasse o seu funcionamento. No subcapítulo que se segue é apresentado esse modelo.

4.2.3. Integração da interface com o componente que modela o seu funcionamento, desenvolvido no ambiente IOPT-Flow Editor

Após o desenho da interface e a configuração correta dos seus parâmetros – que foram continuamente aperfeiçoados – o componente da interface foi integrado com um outro componente, que contém o modelo responsável por afetar a interface e receber informações sobre as ações que o utilizador efetua na mesma. O modelo é apresentado na Figura 4.17 e está disponível no ficheiro "chair_engine.xml".

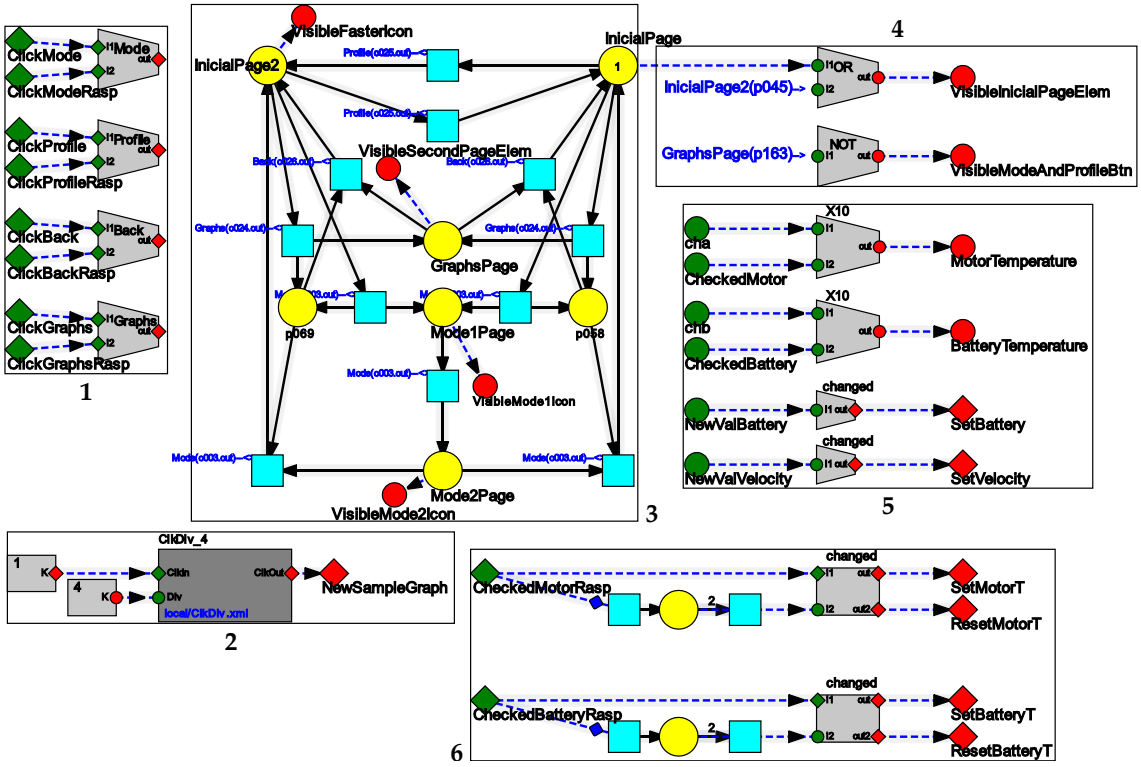


Figura 4.17: Modelo DS-Pnet que modela o funcionamento da interface para cadeiras de rodas elétricas, desenvolvido no IOPT-Flow Editor.

Na Figura 4.17 foram destacadas seis áreas no desenho do modelo:

- Em 1 os componentes cinzentos correspondem a operações "or" entre os respetivos eventos de entrada, que dizem respeito à deteção de um clique num botão da interface e num botão físico, implementado numa montagem que será apresentada mais adiante; os resultados das operações afetam as transições da rede de Petri para permitir mudanças entre as páginas da Figura 4.13;
- Em 2 é criado o evento "NewSampleGraph" que permite a apresentação de novos valores nos gráficos no *scope*;

- Em 3 encontra-se uma rede de Petri onde 5 lugares correspondem às páginas apresentadas na Figura 4.13, e os restantes permitem identificar qual o perfil que está a ser usado pelo utilizador, por forma a garantir o retrocesso para a página inicial correta, dependendo do perfil;
- Também foram estabelecidas ligações de lugares da rede de Petri a sinais de saída do modelo, que permitem definir quais os elementos da interface que ficam visíveis ao utilizador num determinado momento; dois deles são afetados pelas condições apresentadas em 4;
- Em 5 é realizado o controlo dos valores que são enviados à interface, nomeadamente os valores dos gráficos, que devem ser iguais aos sinais "cha" e "chb" apenas quando alguma *checkbox* é selecionada; ou a atualização dos valores da bateria e da velocidade que ocorre quando esse valor altera;
- Em 6 são guardados os estados das *checkboxes* para que consecutivos cliques no botão da montagem permitam alterar o estado de uma *checkbox* para o respetivo complementar.

O componente do modelo da Figura 4.17 é o que se apresenta na Figura 4.18.

Figura 4.18: Componente DS-Pnet do modelo funcional da interface para cadeiras de rodas elétricas, criado no IOPT-Flow Editor.

A Figura 4.19 apresenta-se o modelo onde foram estabelecidas as ligações entre o componente da Figura 4.18 e o componente DS-Pnet gerado automaticamente no ambiente IOPT-Flow GUI Editor; para além disso, o modelo também é composto por um conjunto de 4 sinais e 6 eventos de entrada.

Os sinais e eventos de entrada correspondem à deteção da ocorrência de cliques nos botões de uma montagem que foi realizada com recurso a uma placa Raspberry Pi 3, e que será apresenta mais adiante; e correspondem também aos valores que informam sobre a velocidade, o carregamento da bateria e as medições das suas temperaturas.

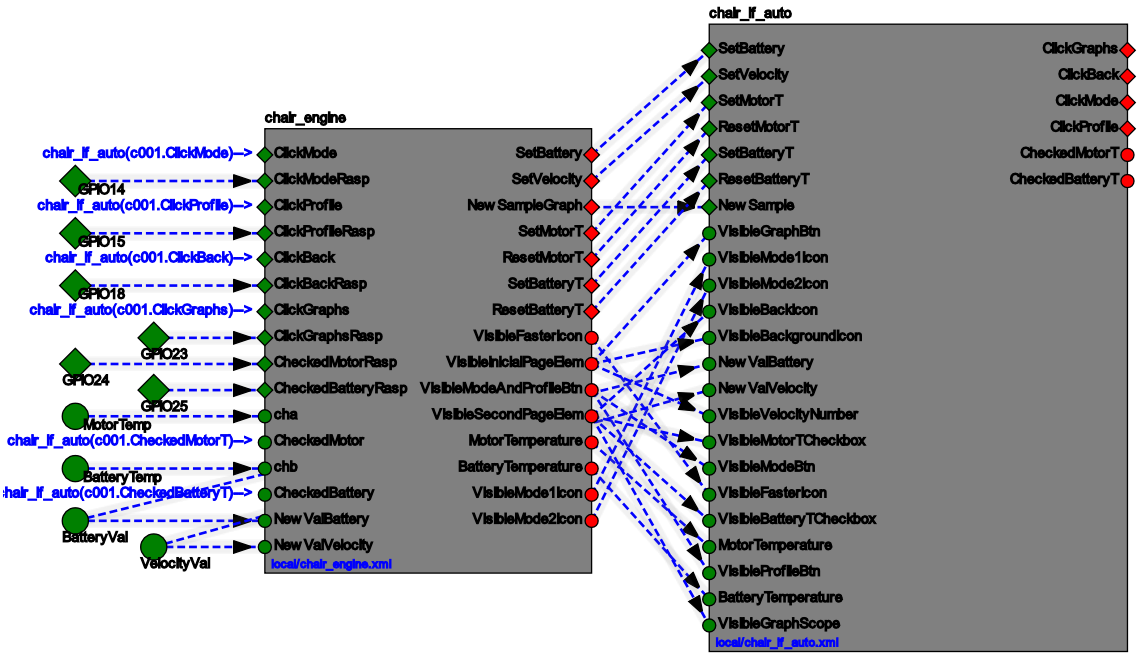


Figura 4.19: Modelo DS-Pnet que junta o componente da interface para cadeiras de rodas elétricas, gerado automaticamente pelo ambiente IOPT-Flow GUI Editor, e o componente que modela o seu funcionamento.

No ambiente IOPT-Flow Editor realizou-se a simulação do modelo do modelo da Figura 4.19, e confirmou-se o funcionamento correto da interface da Figura 4.13. De seguida é apresentado com mais detalhe outro modo usado para validar a correta geração automática de interfaces de utilizador: foi realizada uma montagem incluído o uso da placa Raspberry Pi 3 e realizado a simulação remota do modelo da Figura 4.19.

4.2.4. Simulação remota

Por forma a obter mais resultados sobre a correta implementação e geração de interfaces gráficas de utilizador, procedeu-se à execução e simulação remota do segundo exemplo implementado. A simulação remota do modelo da Figura 4.19 foi realizada procedendo ao uso da ferramenta "remote debugger" e do gerador de código C automático, disponibilizados pelo ambiente IOPT-Flow Editor; e do uso de uma placa Raspberry Pi 3 Modelo B V1.2.

4.2.4.1. Remote debugger

A ferramenta "remote debugger" [29] disponibilizada pelo ambiente IOPT-Flow Editor permite realizar a simulação remota e local de modelos DS-Pnet. Através do "remote debugger" é possível observar e afetar o comportamento de modelos DS-Pnet, cujo código C é compilado e executado por dispositivos embutidos, a comunicarem com o ambiente IOPT-Flow Editor sobre o protocolo de comunicação JSON (*JavaScript Object Notation*) /HTTP.

Como o "remote debugger" permite forçar valores de sinais e eventos do controlador que está em execução, durante os testes efetuados foram alterados os valores das entradas conectadas

ao componente "engine" do modelo da Figura 4.19. Desta forma, foi possível emular valores do carregamento da bateria e da velocidade da cadeira, e das temperaturas medidas e apresentadas no gráfico da interface. Do mesmo modo, através do "remote debugger" é possível visualizar o comportamento de um controlador, aquando da interação direta do utilizador com o dispositivo onde é executado o respetivo código C.

Assim, para realizar a simulação remota do modelo da Figura 4.19, foi necessário garantir que, tanto o Raspberry Pi onde foi compilado e executado o código C do modelo, bem como o computador onde foi aberto o "remote debugger", se encontrassem conectados à mesma rede; depois, no painel do aplicativo apresentado na Figura 4.20, foram preenchidos os parâmetros solicitados, nomeadamente: o "user", a "Password", e o "IP Address" do Raspberry Pi, com porto igual a 9000.

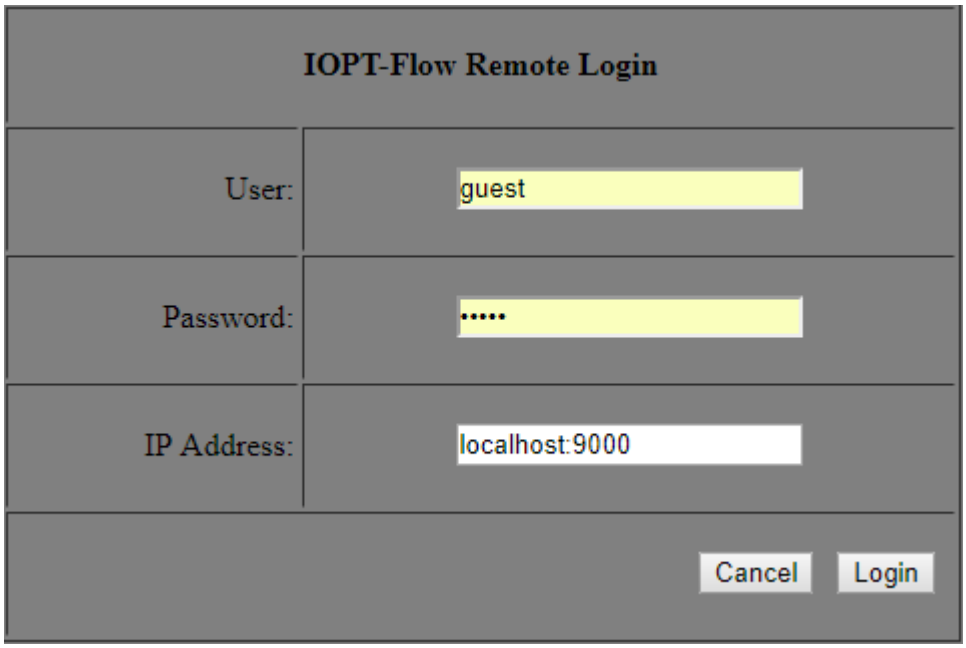


Figura 4.20: Painel inicial do "remote debugger".

De seguida são apresentadas algumas características sobre o gerador de código C automático.

4.2.4.2. Gerador de código C automático

O gerador de código C automático do ambiente IOPT-Flow é muito importante para a implementação de sistemas distribuídos, oferecendo uma estrutura responsável por estabelecer a comunicação com dispositivos remotos através do protocolo de comunicação JSON/HTTP.

Aquando da geração do código C do modelo, são gerados vários ficheiros; destacam-se os seguintes ficheiros: "model_types.h", "model_exec_step.c", "model_maic.c", "model_io.c" e "linux_sys_gpio.c" [29]. Note-se:

- O ficheiro "model_types.h" apresenta a declaração da estrutura de dados do modelo, incluindo entradas e saídas, sinais internos, operações *dataflow*, lugares, transições, entre outros; se o modelo for composto por componentes, é definida uma estrutura para cada um deles, incluindo informações sobre os seus parâmetros, sinais de entrada e saída;
- O ficheiro "model_exec_step.c" contém o código de semântica de execução do modelo;
- No ficheiro "model_main.c" realiza-se a inicialização de valores, o *setup* do *hardware*, e a comunicação remota; para além disso, existe uma função *loop* que chama a função que lê valores de entrada no *hardware*, a função que executa o código de semântica de execução, e outra que escreve os valores de saída; a cada início e fim de um *loop* são processados *requests* e subscrições remotas, respetivamente, sobre o protocolo JSON/HTTP;
- O ficheiro "model_io.c" inicializa os pinos da placa usada, e implementa as funções de leitura e escrita executadas no *loop* do ficheiro "model_main.c"; por sua vez, essas funções fazem uso de funções fornecidas pelo ficheiro "linux_sys_gpio.c"; no final de cada leitura, é executa a função ioptf_applyForcedSignalValues() que altera o valor dos sinais e eventos de entrada do modelo no "remote debugger";
- O ficheiro "linux_sys_gpio.c" executa operações GPIO (*General Purpose Input/Output*), implementando as funções pinMode(), digitalRead(), digitalWrite(), analogRead() e analogWrite().

Para realizar a simulação remota do modelo da Figura 4.19, os ficheiros criados pelo gerador de código C automático foram compilados fazendo uso de uma placa Raspberry Pi 3 Modelo B V1.2.

4.2.4.3. Raspberry Pi 3 Modelo B V1.2

A placa Raspberry Pi 3 Modelo B V1.2 possui um processador de 64 bits de quatro núcleos ARM (*Advanced RISC Machine*) Cortex A53 a 1,2 GHz e uma memória de um 1GB. Salientam-se também as seguintes características: contém um *slot* para cartão de memória, nomeadamente um Micro SD (*Secure Digital Card*), onde se instalou o sistema operativo Raspbian; um módulo de comunicação *wireless*, ideal para realizar a simulação remota; e 40 pinos digitais, dos quais 28 são GPIO.

Seguidamente, apresenta-se a montagem efetuada para conectar um conjunto de botões aos pinos GPIO do Raspberry Pi, possibilitando a afetação do modelo da Figura 4.19, executado pelo dispositivo e apresentado no "remote debugger".

4.2.4.4. Montagem

Como se pode observar na Figura 4.19, e como referido anteriormente, foram adicionados um conjunto de 4 sinais e 6 eventos de entrada ao modelo que integra a interface gráfica de utilizador para cadeiras de rodas elétricas – gerado automaticamente no IOPT-Flow GUI Editor – e o componente que modela o seu funcionamento; são eles, os sinais "MotorTemp", "BatteryTemp", "BatteryVal" e "VelocityVal", e os eventos "GPIO14", "GPIO15", "GPIO18", "GPIO23", "GPIO24" e "GPIO25". Todos os sinais e eventos foram conectados ao componente "engine", e dois dos sinais

também foram ligados às entradas "NewValBattery" e "NewValVelocity" do componente da interface.

Aquando da simulação remota, os valores dos sinais foram definidos e alterados diretamente no ambiente do "remote debugger", apresentado na Figura 4.22; desta forma, foi possível emular valores do carregamento e da velocidade da cadeira, e das temperaturas do motor e da bateria, apresentadas no gráfico da interface. Por outro lado, os valores dos eventos puderam ser alterados tanto pelo "remote debugger", como pela interação do utilizador com botões físicos implementados numa montagem conectada aos pinos do Raspberry Pi, apresentada na Figura 4.21.

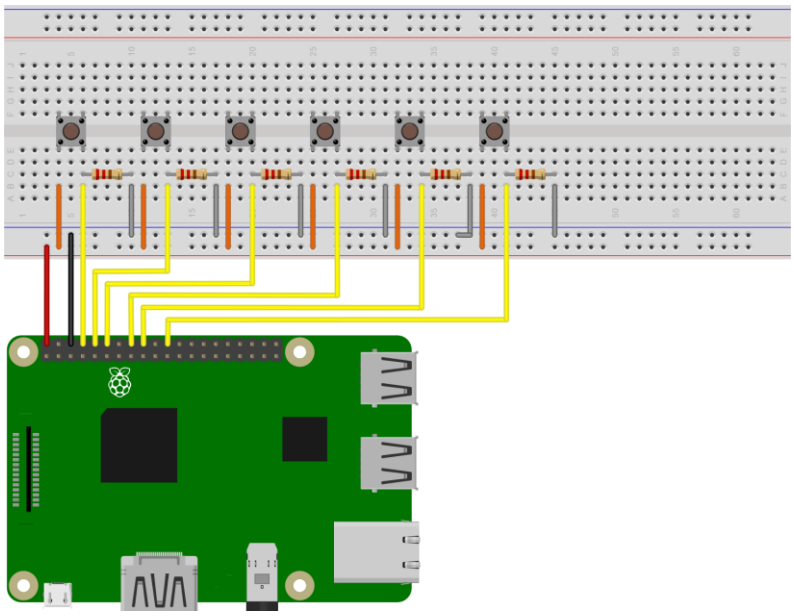


Figura 4.21: Montagem que conecta 6 botões físicos aos pinos GPIO do Raspberry Pi 3 Modelo B V1.2.

A montagem da Figura 4.21 apresenta a conexão de um conjunto de 6 botões ao Raspberry Pi, usado aquando da simulação remota. A *breadboard* e, por conseguinte, os botões, foram alimentados a 5V, e cada um dos botões foi conectado a um pino GPIO da placa.

Assim, cada botão corresponde a um evento de entrada mencionado acima, tal que, o nome de cada evento está relacionado com a ligação efetuada a cada botão. Da esquerda para a direita, os botões foram ligados aos pinos GPIO 14, 15, 18, 23, 24 e 25; e caracterizam o clique nos botões "Mode", "Profile", "Back", "Graphs", e nas *checkboxes* relacionadas com o motor e a bateria, respetivamente.

Efetuada a montagem, procedeu-se à compilação e execução do modelo da Figura 4.19, dando-se início à sua simulação remota através do "remote debugger", estabelecendo-se a sua ligação com o dispositivo. Seguidamente são apresentados os resultados obtidos.

4.2.4.5. Resultados obtidos

Verificada a correta comunicação do dispositivo Raspberry Pi com o "remote debugger", apresentado na Figura 4.22, procedeu-se a um conjunto de testes e validações, enumerados de seguida.

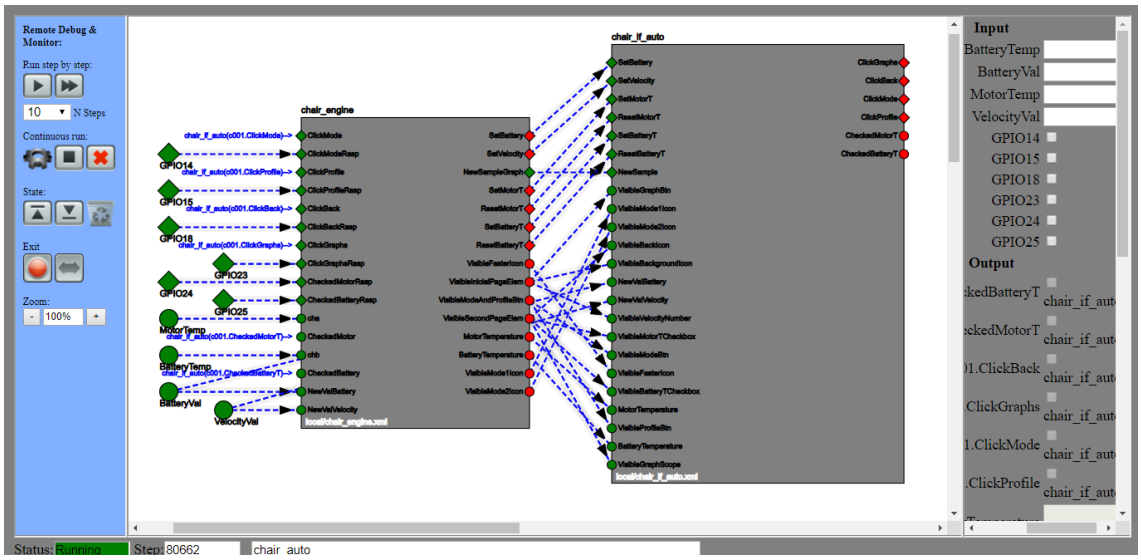


Figura 4.22: "Remote debugger" quando se inicia a simulação e execução remota de um modelo.

1. Inicialização de valores a partir do "remote debugger"

Primeiramente, através do "remote debugger" realizou-se a inicialização dos sinais de entrada do modelo, nomeadamente os sinais "MotorTemp", "BatteryTemp", "BatteryVal" e "VelocityVal", aos quais foram atribuídos os valores 50, 100, 99 e 18, respetivamente; tal atribuição pode ser observada na Figura 4.23 e na Figura 4.24. Na Figura 4.24 observa-se claramente a alteração dos sinais, que passaram a apresentar cor vermelha.

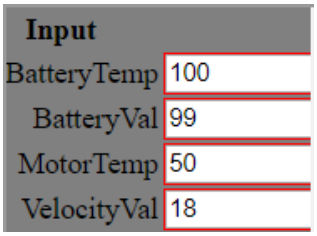


Figura 4.23: Inicialização de sinais de entrada aquando da execução remota do modelo.

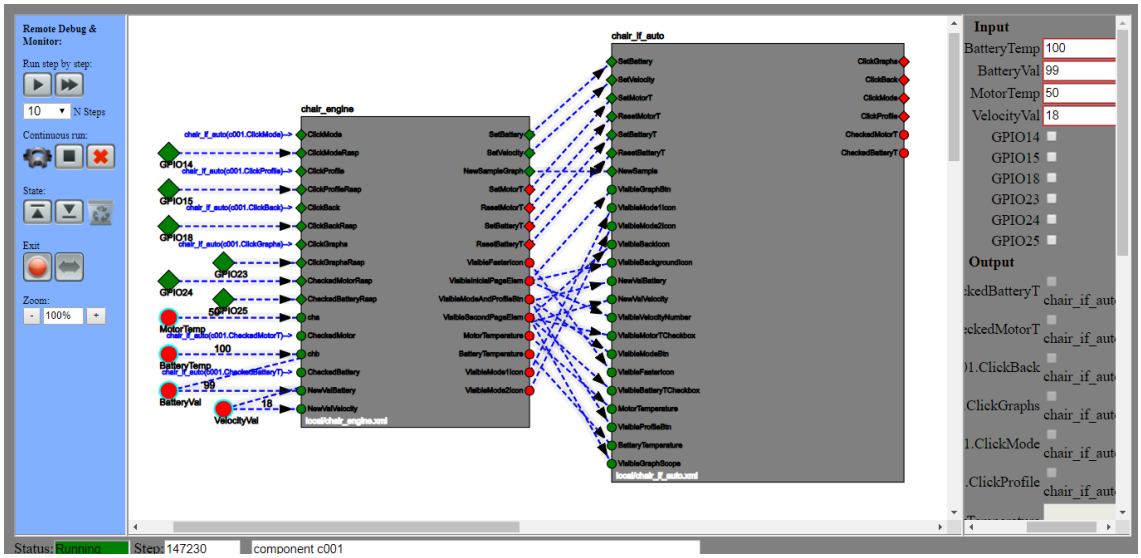
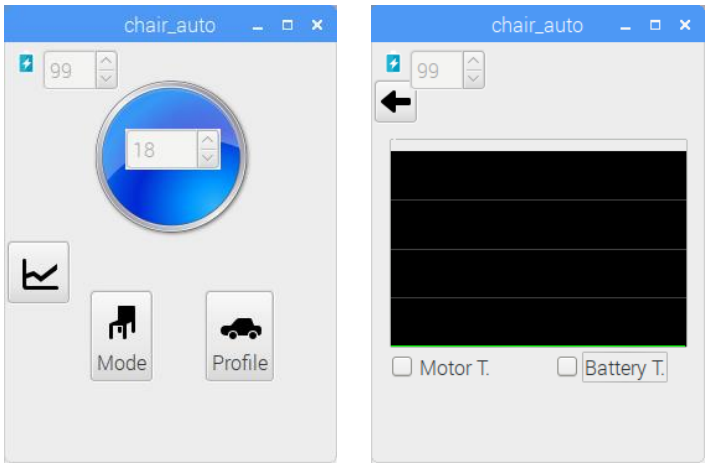


Figura 4.24: "Remote debugger" após a inicialização dos sinais de entrada de um modelo.

Do mesmo modo que se verificou a alteração do modelo DS-Pnet no "remote debugger", a interface do modelo – visualizada através da execução do modelo por parte do Raspberry Pi – passou a apresentar os valores atribuídos aos sinais de entrada. A Figura 4.25 apresenta a página inicial e a página que apresenta os gráficos correspondentes às temperaturas do motor e da bateria, aquando da inicialização dos sinais de entrada. Como esperado, a bateria apresenta o valor 99, e a velocidade tem o valor 18; por outro lado, não são apresentados quaisquer gráficos, visto não terem sido selecionadas nenhuma das *checkboxes*.



(a) Página Inicial

(b) Página para apresentar gráficos

Figura 4.25: Interface para cadeiras de rodas elétricas após a inicialização dos sinais de entrada do modelo, a partir da interação do utilizador com o "remote debugger".

2. Apresentação dos gráficos através da interação com a interface

Depois de confirmada a afetação do modelo e respetiva interface através do "remote debugger", realizou-se um teste para aferir a mesma capacidade, aquando da interação do utilizador com a própria interface; foi solicitada a apresentação dos gráficos das temperaturas, selecionando-se cada uma das *checkboxes*. Tal ação pode ser observada na Figura 4.26.

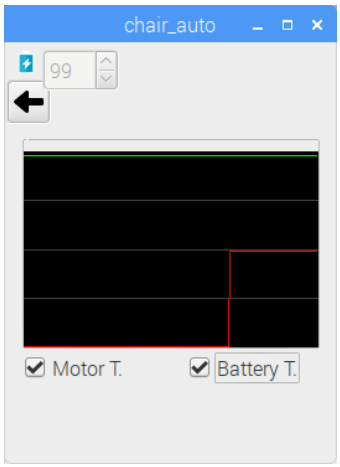


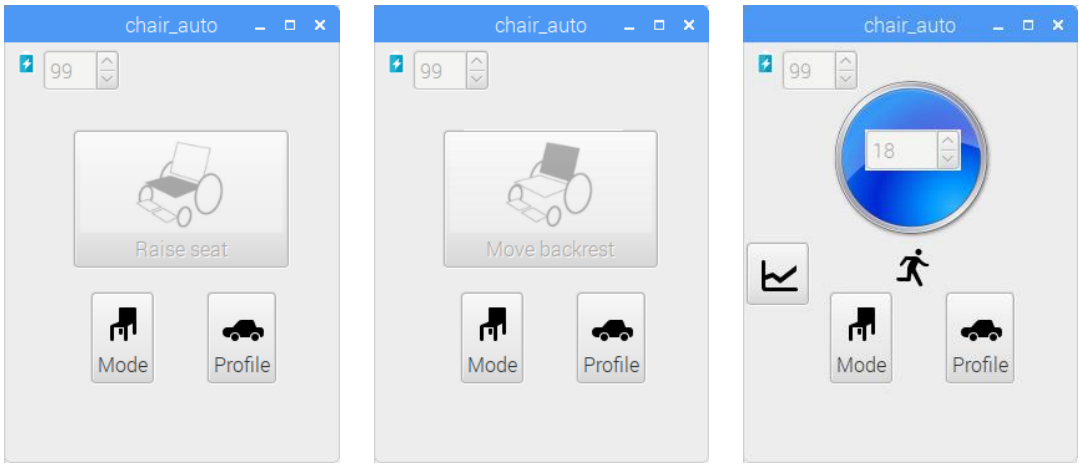
Figura 4.26: Interface para cadeiras de rodas elétricas a apresentar os sinais "MotorTemp" e "Battery-Temp", a partir da interação do utilizador com a interface.

Na Figura 4.26, o gráfico verde corresponde ao valor atribuído ao sinal "BatteryTemp", e o vermelho ao valor atribuído sinal "MotorTemp"; 100 e 50, respetivamente. Assim, percebe-se que estes sinais também foram corretamente inicializados no "remote debugger"; e confirma-se a possibilidade de interação com o modelo através da sua interface.

3. Alteração do modo e perfil de funcionamento da cadeira através dos botões da montagem

O terceiro método de interação do utilizador com o modelo é através da montagem da Figura 4.21. Como explicado anteriormente, cada botão da montagem foi conectado a um pino GPIO do Raspberry Pi.

Pretendendo-se confirmar a resposta do modelo a eventos despoletados por cliques nos botões da montagem, clicou se nos botões físicos representativos dos botões "Mode" e "Profile" da interface, correspondentes aos eventos de entrada "GPIO14" e "GPIO15". Através dessa ação verificou-se a mudança do modo e perfil de funcionamento da cadeira, cujo resultado é apresentado na Figura 4.27.



(a) Modo 1 (b) Modo 2 (c) Perfil 2 ativado

Figura 4.27: Interface para cadeiras de rodas elétricas a apresentar os diferentes modos e perfis de funcionamento, a partir da interação do utilizador com a montagem.

4. **Apresentação dos gráficos através dos botões da montagem e alteração de valores a partir do "remote debugger"**

Após validadas as três formas de interação do utilizador com o modelo, nomeadamente através do "remote debugger", da interface, e de uma montagem; e observado o comportamento do mesmo, através do "remote debugger" e da interface, experimentou-se retornar à página de apresentação dos gráficos através da montagem, e alterar o valor do sinal "BatteryTemp" para 45. Na Figura 4.28 é apresentado o resultado da ação.

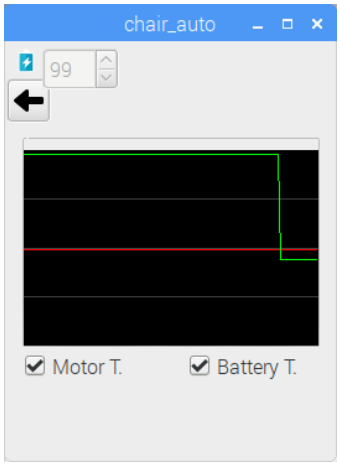


Figura 4.28: Interface para cadeiras de rodas elétricas a apresentar os sinais "MotorTemp" e "BatteryTemp", aquando da alteração do valor do segundo sinal para 45.

Como se pode observar na Figura 4.28, a certa altura verifica-se uma mudança abrupta no gráfico de cor verde, que corresponde à alteração do valor do sinal "BatteryTemp", de 100 para 45; perceba-se que a mudança ocorreu no instante em que foi alterado o valor do sinal no ambiente "remote debugger".

Dos quatro cenários apresentados imediatamente acima, os dois últimos inferem o uso da montagem da Figura 4.21. Afim de confirmar a sua utilização, para a obtenção dos resultados anteriores, apresenta-se a Figura 4.29. Na Figura 4.29 é possível observar a alteração dos estados dos eventos "GPIO18" e "GPIO23", no mesmo instante.

Ora, atente-se ao facto de que, através do "remote debugger" ou da interface, os eventos do modelo podem ser foçados, mas não ao mesmo tempo; nesse caso, para que ocorressem dois eventos num dado instante, seria necessário clicar com o ponteiro do rato em dois sítios aos mesmo tempo. No entanto, através da montagem, isso é possível.

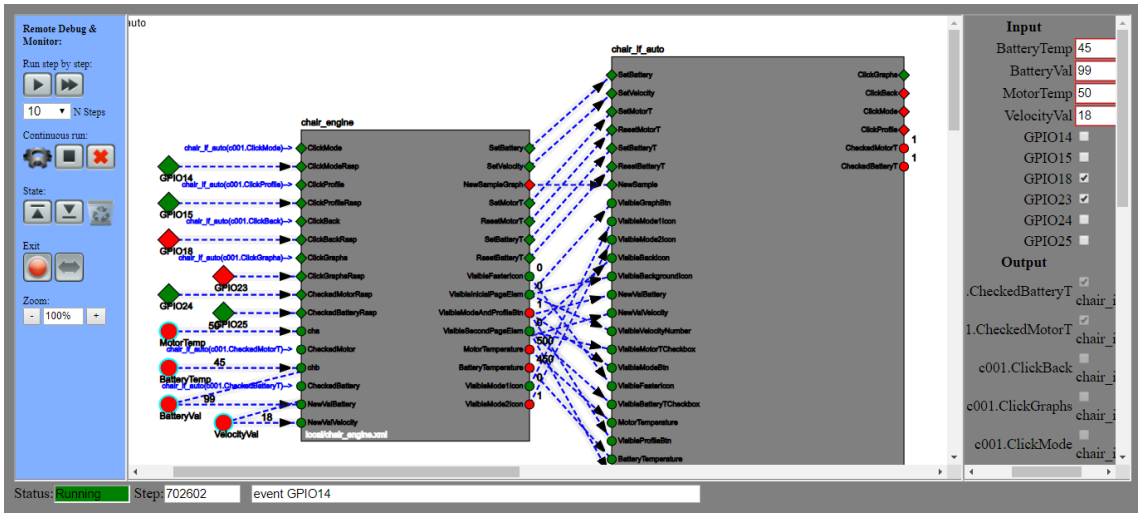


Figura 4.29: "Remote debugger" aquando da ocorrência dos eventos "GPIO18" e "GPIO23", durante a execução remota do modelo.

5

Considerações Finais

O presente, e último capítulo deste documento, apresenta uma reflexão acerca de todo o trabalho que foi desenvolvido nesta tese, destacando a princípio, o sucesso na implementação do editor de interfaces gráficas de utilizador e gerador de componentes IOPT-Flow; o ambiente pode ser acedido em http://gres.uninova.pt/~clo/iopt-flow/inter_clo.php.

Analisando a implementação do novo ambiente, este pode ser considerado um conjunto de duas ferramentas: uma que permite a edição de interfaces gráficas de utilizador com características personalizadas, dependendo da sua aplicação; e outra que permite a conversão dessas interfaces gráficas de utilizador para modelos e componentes IOPT-Flow, que podem ser usados em projetos desenvolvidos no ambiente IOPT-Flow Editor.

Efetivamente, o ambiente IOPT-Flow GUI Editor foi desenvolvido como parte integrante do ambiente IOPT-Flow Editor, por sua vez, dedicado ao desenvolvimento de controladores de sistemas embutidos e ciberfísicos, onde os modelos são desenhados tendo como base o formalismo DS-Pnet, como apresentado na Secção 2.8. Para além disso, este ambiente oferece um conjunto de ferramentas de automatização de projeto, das quais se conseguiu tirar proveito, nomeadamente o simulador, o "remote debugger" e o gerador de código automático.

Comparando o ambiente IOPT-Flow GUI Editor com as ferramentas *Animator* e *Synoptic*, apresentadas na Secção 2.6, atente-se às seguintes ideias. À semelhança do que acontece com o *Animator*, o novo ambiente possibilita a criação de interfaces gráficas de utilizador para sistemas embutidos, sem a necessidade de se escrever qualquer código. De facto, tanto o novo ambiente, como o *Animator* possibilitam a criação de GUI para modelos de redes de Petri IOPT, sendo que o primeiro considera ainda modelos que possam também conter elementos *dataflow*, ou seja, modelos IOPT-Flow.

Outra vantagem do novo ambiente é o facto de não necessitar de saber como é composto o modelo para o qual se pretende criar uma interface. Enquanto o *Animator* necessita de receber o ficheiro PNML, que contém todas as características da rede de Petri IOPT, no ambiente IOPT-Flow GUI Editor, o utilizador pode começar a editar uma nova interface a qualquer momento, sem necessitar de saber como é composto o modelo comportamental da mesma. Neste caso, é

apenas requerido o conhecimento dos sinais e eventos de entrada e saída do modelo IOPT-Flow, nomeadamente o que eles podem representar. Esta característica torna o IOPT-Flow GUI Editor um ambiente independente de qualquer modelo para o qual se esteja a criar uma interface, o que significa que poderá facilmente ser integrado com outras ferramentas de desenvolvimento de sistemas que usem outros formalismos de modelação.

Do mesmo modo, no novo ambiente pode-se criar uma interface gráfica de utilizador, que posteriormente pode ser usada sem que tenham sido definidas quaisquer regras que relacionem as características da GUI, com as características do modelo comportamental para o qual foi criada. Ou seja, enquanto o *Synoptic* precisa de receber informação de 5 ficheiros – Figura 2.5 – para que possa verificar mudanças no modelo, determinar o próximo estado de execução e atualizar automaticamente a GUI, de acordo com as regras definidas; para se usar a interface criada pelo novo ambiente, é apenas necessário que este gere um par de ficheiros XML que contenham o modelo e o componente IOPT-Flow da interface.

Assim, a interface gerada pelo ambiente IOPT-Flow GUI Editor poderá ser executada pelo ambiente IOPT-Flow Editor, sem que seja necessária a preparação que é efetuada no caso do *Synoptic*. Para além disso, após estabelecidas as ligações entre as entradas e saídas do modelo comportamental e a respetiva GUI, pode-se gerar o código C automático, usado para executar o modelo e a GUI num dispositivo remoto, que o compila e executa.

Outra vantagem do novo ambiente é a interface criada permanecer disponível no ambiente IOPT-Flow GUI Edtor, no formato HTML, podendo ser modificada e alterada, sem que isso afete o modelo gerado; que só é alterado se for solicitada a sua conversão.

No capítulo *Testes e Exemplos de Validação* foram apresentadas duas aplicações com finalidades distintas, que permitiram validar o desenho e conversão de uma GUI no novo ambiente: uma interface gráfica de utilizador para controlo de um jogo Pong; e uma interface que pode ser aplicada à visualização e monitorização de parâmetros associados ao funcionamento de uma cadeira de rodas elétrica.

A validação do desenho e da conversão das interfaces passou primeiro por uma fase de análise dos ficheiros XML gerados automaticamente. Depois, os componentes IOPT-Flow das interfaces foram integrados com outros componentes que modelam o seu funcionamento, e foram sujeitos a um conjunto de simulações no ambiente IOPT-Flow Editor; o ambiente IOPT-Flow Editor permite simular local e remotamente modelos DS-Pnet. Permite também monitorizar e controlar sistemas/controladores que estão a correr localmente ou remotamente.

Assim, para validar o desenho e a conversão da interface do jogo Pong, foi realizada a sua simulação local, comparando o seu comportamento com aquele que se observou aquando da simulação do mesmo jogo, com uma interface criada manualmente no ambiente IOPT-Flow Editor; ou seja, comparou-se a interface gerada automaticamente, com uma interface criada manualmente. Os resultados foram positivos, considerando que foi necessário muito pouco tempo para se criar a interface do jogo no novo ambiente; e que, foram garantidas todas as ligações necessárias para conectar os sinais e eventos de entrada e saída aos respetivos componentes IOPT-Flow, apresentado o valor definido aquando da parametrização dos *widgets* correspondentes.

De igual forma, a interface, desenvolvida para visualização e monitorização de parâmetros associados ao funcionamento de uma cadeira de rodas elétrica, foi simulada localmente, obtendo-se os resultados esperados; e posteriormente foi sujeita a uma simulação remota. Para realizar a simulação remota usou-se a ferramenta "remote debugger" e o gerador de código C automático, disponibilizados pelo ambiente IOPT-Flow Editor, e uma placa Raspberry Pi 3 Modelo B V1.2. Os resultados apresentaram-se coerentes com os da simulação local, sendo que foi possível observar o correto funcionamento da interface considerando três modos de interação do utilizador com o modelo: através do aplicativo "remote debugger"; da interface do modelo, cujo código C foi compilado e executado pelo Raspberry Pi; e através de uma montagem conectada a esse dispositivo. O comportamento do modelo foi validado através do "remote debugger" e dos valores apresentados na interface.

Através dos resultados obtidos conseguimos validar o ambiente que foi desenvolvido; e conseguimos transpor o uso do ambiente IOPT-Flow GUI Editor para outro tipo de aplicações e indústrias, e incluir a sua utilização a nível académico, apoiando o desenho de GUI para projetos que sejam desenvolvidos no ambiente IOPT-Flow Editor. De seguida apresenta-se alguns pontos que deverão ser considerados para trabalhos futuros.

Como trabalhos futuros considere-se:

- O aperfeiçoamento do novo ambiente, não só para melhorar a experiência do utilizador, mas também a nível das ferramentas oferecidas, permitindo a geração de interfaces para serem usadas por outros ambientes, como acontece com o ambiente IOPT-Flow Editor; essas ferramentas poderão converter as interfaces para outros formalismos e linguagens de desenvolvimento;
- Também se propõe a criação de interfaces a partir do conhecimento que se tenha sobre um modelo "engine", nomeadamente sobre os sinais e eventos que caracterizam o modelo de funcionamento de algum sistema; os sinais de saída e entrada do modelo do sistema deverão ser diretamente encaminhados ao IOPT-Flow GUI Editor informando o utilizador sobre quais serão os sinais e eventos que estarão ao seu dispor para interagir com o sistema;
- Do mesmo modo, propõe-se o melhoramento da experiência na elaboração de GUI, permitindo a execução de uma interface em tempo real, tanto no ambiente IOPT-Flow GUI Editor, como no ambiente IOPT-Flow Editor, à semelhança do que acontece com outros programas e ferramentas apresentadas no estado de arte.

Por fim, aquando da realização deste trabalho, foi publicado um artigo inserido na conferência "REC 2018 - XIV Jornadas sobre Sistemas Reconfiguráveis", que se realizou em fevereiro de 2018 na FCT; o artigo publicado, "Monitorização de Cadeiras de Rodas através de Ferramentas de Automatização de Projeto" , focou o uso ferramentas de automatização para a implementação de um sistema monitor dos parâmetros associados ao funcionamento de uma cadeira de rodas elétrica.

Referências Bibliográficas

[1] F. Pereira and L. Gomes, "The IOPT-Flow framework pairing Petri nets and data-flows for embedded controller development", IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society, Florence, 2016, pp. 4832-4837.

[2] F. Pereira and L. Gomes, "Combining data-flows and petri nets for cyber-physical systems specification", in Technological Innovation for Cyber-Physical Systems - 7th IFIP WG 5.5/SOCOLNET Advanced Doctoral Conference on Computing, Electrical and Industrial Systems, DoCEIS 2016, Proceedings, vol. 470, IFIP Advances in Information and Communication Technology, 2016, pp. 65-76.

[3] L. Gomes and J. Lourenco, "Rapid Prototyping of Graphical User Interfaces for Petri-Net-Based Controllers", in IEEE Transactions on Industrial Electronics, vol. 57, no. 5, pp. 1806-1813, May 2010.

[4] L. Gomes and J. Lourenco, "Petri nets-based automatic generation GUI tools for embedded systems," 2008 Conference on Human System Interactions, Krakow, 2008, pp. 269-274.

[5] M. Erwig, K. Smeltzer, and X. Wang, “What is a visual language?”, in Journal of Visual Languages & Computing, vol. 38, pp. 9–17, 2017.

[6] D. Draheim, C. Lutteroth, and G. Weber, “Graphical user interfaces as documents”, in Proceedings of the 7th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: design centered HCI (CHINZ '06), ACM, New York, NY, USA, pp. 67-74, 2006.

[7] J. Bishop and N. Horspool, "Cross-Platform Development: Software that Lasts", in Computer, vol. 39, no. 10, pp. 26-35, Oct. 2006.

[8] R. V Roque, “OpenBlocks: an extendable framework for graphical block programming systems”, MSc. Thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 2007.

[9] “Open Blocks Download Page | MIT STEP.” [Online]. Available: http://web.mit.edu/mitstep/openblocks.html. [Accessed: 02-Mar-2018].

[10] H. Lin, C. Tu, and Y. Hwang, “CUDABlock: A GUI Programming Tool for CUDA”, 2015 44th International Conference on Parallel Processing Workshops, Beijing, 2015, pp. 37-42.

[11] K. B. Chavhan and R. T. Ugale, "Automated test bench for an induction motor using LabVIEW", 2016 IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES), Delhi, 2016, pp. 1-6.

[12] E. Vavilina and G. Gaigals, "Improved LabVIEW code generation", 2015 IEEE 3rd Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE), Riga, 2015, pp. 1-4.

[13] "Tutorial: Block Diagram - National Instruments." [Online]. Available: http://www.ni.com/tutorial/7565/en/. [Accessed: 25-Feb-2018].

[14] S. D. Rathod, "Automatic code generation with business logic by capturing attributes from user interface via XML", 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT), Chennai, 2016, pp. 1480-1484.

[15] T. H. Y. Ling, L. J. Wong, J. E. H. Tan and C. K. Lee, "XBee Wireless Blood Pressure Monitoring System with Microsoft Visual Studio Computer Interfacing", 2015 6th International Conference on Intelligent Systems, Modelling and Simulation, Kuala Lumpur, 2015, pp. 5-9.

[16] "Form Class (System.Windows.Forms)." [Online]. Available: https://msdn.microsoft.com/en-us/library/system.windows.forms.form(v=vs.110).aspx#Explicit Interface Implementations. [Accessed: 25-Feb-2018].

[17] "WindowBuilder." [Online]. Available: http://www.eclipse.org/proposals/tools.windowbuilder/. [Accessed: 02-Mar-2018].

[18] "WindowBuilder." [Online]. Available: https://www.eclipse.org/windowbuilder/. [Accessed: 02-Mar-2018].

[19] J. Lourenco and L. Gomes, "Animated Graphical User Interface Generator Framework for Input-Output Place-Transition Petri Net Models", In van Hee K.M., Valk R. (eds) Applications and Theory of Petri Nets, PETRI NETS 2008, Lecture Notes in Computer Science, vol 5062, pp. 409–418, Springer, Berlin, Heidelberg.

[20] L. Gomes, F. Moutinho, F. Pereira, J. Ribeiro, A. Costa and J. Barros, "Extending input-output place-transition Petri nets for distributed controller systems development", 2014 International Conference on Mechatronics and Control (ICMC), Jinzhou, 2014, pp. 1099-1104.

[21] L. Gomes, J. P. Barros, A. Costa, and R. Nunes, "The Input-Output Place-Transition Petri Net Class and Associated Tools", in *2007 5th IEEE International Conference on Industrial Informatics*, 2007, vol. 1, pp. 509–514.

[22] F. Moutinho and L. Gomes, "From models to controllers integrating graphical animation in FPGA through automatic code generation", 2009 IEEE International Symposium on Industrial Electronics, Seoul, 2009, pp. 712-717.

[23] F. Pereira, F. Moutinho and L. Gomes, "IOPT-tools — Towards cloud design automation of digital controllers with Petri nets", 2014 International Conference on Mechatronics and Control (ICMC), Jinzhou, 2014, pp. 2414-2419.

[24] R. Campos-Rebelo, F. Pereira, F. Moutinho and L. Gomes, "From IOPT Petri nets to C: An automatic code generator tool", 2011 9th IEEE International Conference on Industrial Informatics, Caparica, Lisbon, 2011, pp. 390-395.

[25] L. Gomes, R. Rebelo, J. P. Barros, A. Costa and R. Pais, "From Petri net models to C implementation of digital controllers", 2010 IEEE International Symposium on Industrial Electronics, Bari, 2010, pp. 3057-3062.

[26] F. Pereira and L. Gomes, "Automatic synthesis of VHDL hardware components from IOPT Petri net models", IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society, Vienna, 2013, pp. 2214-2219.

[27] J. Lourenço, "Modelos comportamentais e redes de Petri na geração automática de animação de sinópticos", MSc. Thesis, FCT-UNL, Dept. de Engenharia Electrotécnica, Lisbon, 2008.

[28] H. Hilal and A. Nangim, "Network security analysis SCADA system automation on industrial process," 2017 International Conference on Broadband Communication, Wireless Sensors and Powering (BCWSP), Jakarta, 2017, pp. 1-6.

[29] F. Pereira, "The DS-Pnet modeling formalism for cyber-physical system development", PhD. Thesis, FCT-UNL, Dept. de Engenharia Electrotécnica, Lisbon, 2017.

[30] C. Lagartinho-Oliveira, R. Campos-Rebelo and F. Moutinho, "Monitorização de Cadeiras de Rodas através de Ferramentas de Automatização de Projeto", in REC'2018 - XIV Jornadas sobre Sistemas Reconfiguráveis, 16-17 Fevereiro de 2018, Monte de Caparica, Portugal.

[31] F. Pereira and L. Gomes, "The IOPT-Flow Modeling Framework Applied to Power Electronics Controllers," in IEEE Transactions on Industrial Electronics, vol. 64, no. 3, pp. 2363-2372, March 2017.

[32] "Electric Wheelchair Market: Global Top Industry Trends & Segments Forecast 2016–2024 - Press Release - Digital Journal." [Online]. Available: http://www.digitaljournal.com/pr/3551838. [Accessed: 25-Feb-2018].

[33] "Invacare LiNX." [Online]. Available: http://www.invacare.es/es/electrónica-linx-ma-p39w26um33es. [Accessed: 25-Feb-2018].

Anexos

Seguidamente, apresenta-se um conjunto de anexos com o código da representação XML do modelo e componente IOPT-Flow, gerados automaticamente no novo ambiente, aquando da criação da interface gráfica da utilizador para o jogo Pong, apresentada no Capítulo 4.

A. Modelo IOPT-Flow da interface do jogo pong, gerado automaticamente a partir da edição e conversão da interface no ambiente IOPT-Flow GUI Editor.

```
<?xml version="1.0"?>
<net name="pong_if_auto.html" type="iopt-flow">
 <component id="i006" class="ui/label_icon.xml" x="170" y="200" width="150" height="140"
rot="0" implementation="iopt-flow" target="external" res_location="./files/gui/ui/
pong_if_auto/ball.png" param_string="-">
   <name off_x="-75" off_y="-75" text="ui_label_icon_ball"/>
   <source_model file="files/ui_label.xml"/>
   <input id="i006.Visible" name="Visible" off_x="-75" off_y="-50" type="boolean"/>
   <input id="i006.X" name="X" off_x="-75" off_y="-30" type="range" min="0" max="1023"/>
   <input id="i006.Y" name="Y" off_x="-75" off_y="-10" type="range" min="0" max="1023"/>
   <input id="i006.Width" name="Width" off_x="-75" off_y="10" type="range" min="0"
max="1023"/>
   <input id="i006.Height" name="Height" off_x="-75" off_y="30" type="range" min="0"
max="1023"/>
   <input id="i006.PageNr" name="PageNr" off_x="-75" off_y="50" type="range" min="0"
max="31"/>
   <output id="i006.Show" name="Show" type="boolean" off_x="75" off_y="-50"/>
   <comment text="Ball" off_x="0" off_y="20"/>
 </component>
 <signal id="RunGame" x="30" y="110" mode="input" type="boolean" min="0" max="1" value="0"
dynamic="none" frac="0"/>
 <arc id="a001" type="read" source="RunGame" target="i006.Visible"/>
 <signal id="X" x="30" y="145" mode="input" type="range" min="-1" max="801" value="0"
dynamic="none" frac="0"/>
 <arc id="a002" type="read" source="X" target="i006.X"/>
 <signal id="Y" x="30" y="180" mode="input" type="range" min="-1" max="801" value="0"
dynamic="none" frac="0"/>
 <arc id="a003" type="read" source="Y" target="i006.Y"/>
 <component id="i007" class="ui/label_icon.xml" x="540" y="200" width="150" height="140"
rot="0" implementation="iopt-flow" target="external" res_location="./files/gui/ui/
pong_if_auto/base_ball.png" param_string="-">
   <name off_x="-75" off_y="-75" text="ui_label_icon_base"/>
   <source_model file="files/ui_label.xml"/>
   <input id="i007.Visible" name="Visible" off_x="-75" off_y="-50" type="boolean"/>
   <input id="i007.X" name="X" off_x="-75" off_y="-30" type="range" min="0" max="1023"/>
   <input id="i007.Y" name="Y" off_x="-75" off_y="-10" type="range" min="0" max="1023"/>
```

```
   <input id="i007.Width" name="Width" off_x="-75" off_y="10" type="range" min="0"
max="1023"/>
   <input id="i007.Height" name="Height" off_x="-75" off_y="30" type="range" min="0"
max="1023"/>
   <input id="i007.PageNr" name="PageNr" off_x="-75" off_y="50" type="range" min="0"
max="31"/>
   <output id="i007.Show" name="Show" type="boolean" off_x="75" off_y="-50"/>
   <comment text="Image fot test!" off_x="0" off_y="20"/>
  </component>
  <signal id="RunGame2" x="400" y="110" mode="input" type="boolean" min="0" max="1" value="0"
dynamic="none" frac="0"/>
  <arc id="a004" type="read" source="RunGame2" target="i007.Visible"/>
  <signal id="BaseX" x="400" y="145" mode="input" type="range" min="-1" max="800" value="400"
dynamic="none" frac="0"/>
  <arc id="a005" type="read" source="BaseX" target="i007.X"/>
  <operation id="o006" x="400" y="180" constant="560" rot="0" shape="rect" size="20"
locked="true">
   <name off_x="-15" off_y="-5" text="560"/>
   <output off_x="20" off_y="0" name="K" id="o006.K" type="range" min="560" max="560"
dynamic="none" frac="0">
    <expression>
     <text>560</text>
     <operand type="literal" value="560"/>
  </expression>
   </output>
  </operation>
  <arc id="a007" type="read" source="o006.K" target="i007.Y"/>
  <component id="b001" class="ui/button.xml" x="910" y="200" width="160" height="160" rot="0"
implementation="iopt-flow" target="external" res_location="./files/gui/ui/pong_if_auto/
ball.png" param_string="@ @">
   <name off_x="-80" off_y="-85" text="ui_start_button"/>
   <source_model file="files/ui_button.xml"/>
   <input id="b001.Visible" name="Visible" off_x="-80" off_y="-60" type="boolean"/>
   <input id="b001.Sensitive" name="Sensitive" off_x="-80" off_y="-40" type="boolean"/>
   <input id="b001.X" name="X" off_x="-80" off_y="-20" type="range" min="0" max="1023"/>
   <input id="b001.Y" name="Y" off_x="-80" off_y="0" type="range" min="0" max="1023"/>
   <input id="b001.Width" name="Width" off_x="-80" off_y="20" type="range" min="0"
max="1023"/>
   <input id="b001.Height" name="Height" off_x="-80" off_y="40" type="range" min="0"
max="1023"/>
   <input id="b001.PageNr" name="PageNr" off_x="-80" off_y="60" type="range" min="0"
max="31"/>
   <output id="b001.Click" name="Click" type="event" off_x="80" off_y="-60"/>
   <output id="b001.Pressed" name="Pressed" off_x="80" off_y="-40" type="boolean"/>
   <comment text="Start Game!" off_x="0" off_y="20"/>
  </component>
  <signal id="ShowStartBtn" x="770" y="95" mode="input" type="boolean" min="0" max="1"
value="0" dynamic="none" frac="0"/>
  <arc id="a008" type="read" source="ShowStartBtn" target="b001.Visible"/>
  <operation id="o009" x="770" y="130" constant="1" rot="0" shape="rect" size="20"
locked="true">
   <name off_x="-15" off_y="-5" text="1"/>
   <output off_x="20" off_y="0" name="K" id="o009.K" type="range" min="1" max="1"
dynamic="none" frac="0">
    <expression>
     <text>1</text>
     <operand type="literal" value="1"/>
    </expression>
   </output>
  </operation>
  <arc id="a010" type="read" source="o009.K" target="b001.Sensitive"/>
  <operation id="o011" x="770" y="165" constant="300" rot="0" shape="rect" size="20"
locked="true">
   <name off_x="-15" off_y="-5" text="300"/>
   <output off_x="20" off_y="0" name="K" id="o011.K" type="range" min="300" max="300"
dynamic="none" frac="0">
    <expression>
     <text>300</text>
     <operand type="literal" value="300"/>
    </expression>
   </output>
  </operation>
  <arc id="a012" type="read" source="o011.K" target="b001.X"/>
  <operation id="o013" x="770" y="200" constant="280" rot="0" shape="rect" size="20"
locked="true">
   <name off_x="-15" off_y="-5" text="280"/>
   <output off_x="20" off_y="0" name="K" id="o013.K" type="range" min="280" max="280"
dynamic="none" frac="0">
```

```
    <expression>
     <text>280</text>
     <operand type="literal" value="280"/>
    </expression>
   </output>
 </operation>
 <arc id="a014" type="read" source="o013.K" target="b001.Y"/>
 <event id="StartBtn" x="1050" y="140" mode="output" io_pin="0">
  <comment text="" off_x="0" off_y="20"/>
 </event>
 <arc id="a015" type="read" source="b001.Click" target="StartBtn"/>
 <component id="b002" class="ui/button.xml" x="170" y="600" width="160" height="160" rot="0"
implementation="iopt-flow" target="external" res_location="./files/gui/ui/pong_if_auto/
end.jpg" param_string="@C[@">
   <name off_x="-80" off_y="-85" text="ui_game_over_button"/>
   <source_model file="files/ui_button.xml"/>
   <input id="b002.Visible" name="Visible" off_x="-80" off_y="-60" type="boolean"/>
   <input id="b002.Sensitive" name="Sensitive" off_x="-80" off_y="-40" type="boolean"/>
   <input id="b002.X" name="X" off_x="-80" off_y="-20" type="range" min="0" max="1023"/>
   <input id="b002.Y" name="Y" off_x="-80" off_y="0" type="range" min="0" max="1023"/>
   <input id="b002.Width" name="Width" off_x="-80" off_y="20" type="range" min="0"
max="1023"/>
   <input id="b002.Height" name="Height" off_x="-80" off_y="40" type="range" min="0"
max="1023"/>
   <input id="b002.PageNr" name="PageNr" off_x="-80" off_y="60" type="range" min="0"
max="31"/>
   <output id="b002.Click" name="Click" type="event" off_x="80" off_y="-60"/>
   <output id="b002.Pressed" name="Pressed" off_x="80" off_y="-40" type="boolean"/>
   <comment text="GAME OVER" off_x="0" off_y="20"/>
 </component>
 <signal id="ShowGameOver" x="30" y="495" mode="input" type="boolean" min="0" max="1"
value="0" dynamic="none" frac="0"/>
 <arc id="a016" type="read" source="ShowGameOver" target="b002.Visible"/>
 <operation id="o017" x="30" y="530" constant="1" rot="0" shape="rect" size="20"
locked="true">
   <name off_x="-15" off_y="-5" text="1"/>
   <output off_x="20" off_y="0" name="K" id="o017.K" type="range" min="1" max="1"
dynamic="none" frac="0">
    <expression>
     <text>1</text>
     <operand type="literal" value="1"/>
    </expression>
   </output>
 </operation>
 <arc id="a018" type="read" source="o017.K" target="b002.Sensitive"/>
 <operation id="o019" x="30" y="565" constant="300" rot="0" shape="rect" size="20"
locked="true">
   <name off_x="-15" off_y="-5" text="300"/>
   <output off_x="20" off_y="0" name="K" id="o019.K" type="range" min="300" max="300"
dynamic="none" frac="0">
    <expression>
     <text>300</text>
     <operand type="literal" value="300"/>
    </expression>
   </output>
 </operation>
 <arc id="a020" type="read" source="o019.K" target="b002.X"/>
 <operation id="o021" x="30" y="600" constant="280" rot="0" shape="rect" size="20"
locked="true">
   <name off_x="-15" off_y="-5" text="280"/>
   <output off_x="20" off_y="0" name="K" id="o021.K" type="range" min="280" max="280"
dynamic="none" frac="0">
    <expression>
      <text>280</text>
      <operand type="literal" value="280"/>
    </expression>
   </output>
 </operation>
 <arc id="a022" type="read" source="o021.K" target="b002.Y"/>
 <event id="ClrGameOver" x="310" y="540" mode="output" io_pin="0">
  <comment text="" off_x="0" off_y="20"/>
 </event>
 <arc id="a023" type="read" source="b002.Click" target="ClrGameOver"/>
 <component id="b003" class="ui/button.xml" x="540" y="600" width="160" height="160" rot="0"
implementation="iopt-flow" target="external" res_location="./files/gui/ui/pong_if_auto/
left.png" param_string="@a@">
   <name off_x="-80" off_y="-85" text="ui_button_left"/>
   <source_model file="files/ui_button.xml"/>
```

```
   <input id="b003.Visible" name="Visible" off_x="-80" off_y="-60" type="boolean"/>
   <input id="b003.Sensitive" name="Sensitive" off_x="-80" off_y="-40" type="boolean"/>
   <input id="b003.X" name="X" off_x="-80" off_y="-20" type="range" min="0" max="1023"/>
   <input id="b003.Y" name="Y" off_x="-80" off_y="0" type="range" min="0" max="1023"/>
   <input id="b003.Width" name="Width" off_x="-80" off_y="20" type="range" min="0"
max="1023"/>
   <input id="b003.Height" name="Height" off_x="-80" off_y="40" type="range" min="0"
max="1023"/>
   <input id="b003.PageNr" name="PageNr" off_x="-80" off_y="60" type="range" min="0"
max="31"/>
   <output id="b003.Click" name="Click" type="event" off_x="80" off_y="-60"/>
   <output id="b003.Pressed" name="Pressed" off_x="80" off_y="-40" type="boolean"/>
   <comment text="" off_x="0" off_y="20"/>
  </component>
  <operation id="o024" x="400" y="495" constant="1" rot="0" shape="rect" size="20"
locked="true">
   <name off_x="-15" off_y="-5" text="1"/>
   <output off_x="20" off_y="0" name="K" id="o024.K" type="range" min="1" max="1"
dynamic="none" frac="0">
    <expression>
     <text>1</text>
     <operand type="literal" value="1"/>
    </expression>
   </output>
  </operation>
  <arc id="a025" type="read" source="o024.K" target="b003.Visible"/>
  <operation id="o026" x="400" y="530" constant="1" rot="0" shape="rect" size="20"
locked="true">
   <name off_x="-15" off_y="-5" text="1"/>
   <output off_x="20" off_y="0" name="K" id="o026.K" type="range" min="1" max="1"
dynamic="none" frac="0">
    <expression>
     <text>1</text>
     <operand type="literal" value="1"/>
    </expression>
   </output>
  </operation>
  <arc id="a027" type="read" source="o026.K" target="b003.Sensitive"/>
  <operation id="o028" x="400" y="565" constant="350" rot="0" shape="rect" size="20"
locked="true">
   <name off_x="-15" off_y="-5" text="350"/>
   <output off_x="20" off_y="0" name="K" id="o028.K" type="range" min="350" max="350"
dynamic="none" frac="0">
    <expression>
     <text>350</text>
     <operand type="literal" value="350"/>
    </expression>
   </output>
  </operation>
  <arc id="a029" type="read" source="o028.K" target="b003.X"/>
  <operation id="o030" x="400" y="600" constant="640" rot="0" shape="rect" size="20"
locked="true">
   <name off_x="-15" off_y="-5" text="640"/>
   <output off_x="20" off_y="0" name="K" id="o030.K" type="range" min="640" max="640"
dynamic="none" frac="0">
    <expression>
     <text>640</text>
     <operand type="literal" value="640"/>
    </expression>
   </output>
  </operation>
  <arc id="a031" type="read" source="o030.K" target="b003.Y"/>
  <signal id="LeftBtn" x="655" y="560" mode="output" type="boolean" min="0" max="1" value="0"
dynamic="none" frac="0"/>
  <arc id="a032" type="read" source="b003.Pressed" target="LeftBtn"/>
  <component id="b004" class="ui/button.xml" x="910" y="600" width="160" height="160" rot="0"
implementation="iopt-flow" target="external" res_location="./files/gui/ui/pong_if_auto/
right.png" param_string="@s@">
   <name off_x="-80" off_y="-85" text="ui_button_right"/>
   <source_model file="files/ui_button.xml"/>
   <input id="b004.Visible" name="Visible" off_x="-80" off_y="-60" type="boolean"/>
   <input id="b004.Sensitive" name="Sensitive" off_x="-80" off_y="-40" type="boolean"/>
   <input id="b004.X" name="X" off_x="-80" off_y="-20" type="range" min="0" max="1023"/>
   <input id="b004.Y" name="Y" off_x="-80" off_y="0" type="range" min="0" max="1023"/>
   <input id="b004.Width" name="Width" off_x="-80" off_y="20" type="range" min="0"
max="1023"/>
   <input id="b004.Height" name="Height" off_x="-80" off_y="40" type="range" min="0"
max="1023"/>
```

```
   <input id="b004.PageNr" name="PageNr" off_x="-80" off_y="60" type="range" min="0"
max="31"/>
   <output id="b004.Click" name="Click" type="event" off_x="80" off_y="-60"/>
   <output id="b004.Pressed" name="Pressed" off_x="80" off_y="-40" type="boolean"/>
   <comment text="" off_x="0" off_y="20"/>
  </component>
  <operation id="o033" x="770" y="495" constant="1" rot="0" shape="rect" size="20"
locked="true">
   <name off_x="-15" off_y="-5" text="1"/>
   <output off_x="20" off_y="0" name="K" id="o033.K" type="range" min="1" max="1"
dynamic="none" frac="0">
    <expression>
     <text>1</text>
     <operand type="literal" value="1"/>
    </expression>
   </output>
  </operation>
  <arc id="a034" type="read" source="o033.K" target="b004.Visible"/>
  <operation id="o035" x="770" y="530" constant="1" rot="0" shape="rect" size="20"
locked="true">
   <name off_x="-15" off_y="-5" text="1"/>
   <output off_x="20" off_y="0" name="K" id="o035.K" type="range" min="1" max="1"
dynamic="none" frac="0">
    <expression>
     <text>1</text>
     <operand type="literal" value="1"/>
    </expression>
   </output>
  </operation>
  <arc id="a036" type="read" source="o035.K" target="b004.Sensitive"/>
  <operation id="o037" x="770" y="565" constant="450" rot="0" shape="rect" size="20"
locked="true">
   <name off_x="-15" off_y="-5" text="450"/>
   <output off_x="20" off_y="0" name="K" id="o037.K" type="range" min="450" max="450"
dynamic="none" frac="0">
    <expression>
     <text>450</text>
     <operand type="literal" value="450"/>
    </expression>
   </output>
  </operation>
  <arc id="a038" type="read" source="o037.K" target="b004.X"/>
  <operation id="o039" x="770" y="600" constant="640" rot="0" shape="rect" size="20"
locked="true">
   <name off_x="-15" off_y="-5" text="640"/>
   <output off_x="20" off_y="0" name="K" id="o039.K" type="range" min="640" max="640"
dynamic="none" frac="0">
    <expression>
     <text>640</text>
     <operand type="literal" value="640"/>
    </expression>
   </output>
  </operation>
  <arc id="a040" type="read" source="o039.K" target="b004.Y"/>
  <signal id="RightBtn" x="1025" y="560" mode="output" type="boolean" min="0" max="1"
value="0" dynamic="none" frac="0"/>
  <arc id="a041" type="read" source="b004.Pressed" target="RightBtn"/>
  <component id="n008" class="ui/number.xml" x="170" y="1000" width="200" height="240" rot="0"
implementation="iopt-flow" target="external" res_location="-" param_string="-">
   <name off_x="-100" off_y="-125" text="ui_number_score"/>
   <source_model file="files/ui_number.xml"/>
   <input id="n008.Set" name="Set" off_x="-100" off_y="-100" type="event"/>
   <input id="n008.NewVal" name="NewVal" off_x="-100" off_y="-80" type="range" min="0"
max="65535"/>
   <input id="n008.Visible" name="Visible" off_x="-100" off_y="-60" type="boolean"/>
   <input id="n008.Sensitive" name="Sensitive" off_x="-100" off_y="-40" type="boolean"/>
   <input id="n008.X" name="X" off_x="-100" off_y="-20" type="range" min="0" max="1023"/>
   <input id="n008.Y" name="Y" off_x="-100" off_y="0" type="range" min="0" max="1023"/>
   <input id="n008.Width" name="Width" off_x="-100" off_y="20" type="range" min="0"
max="1023"/>
   <input id="n008.Height" name="Height" off_x="-100" off_y="40" type="range" min="0"
max="1023"/>
   <input id="n008.PageNr" name="PageNr" off_x="-100" off_y="60" type="range" min="0"
max="31"/>
   <input id="n008.Min" name="Min" off_x="-100" off_y="80" type="range" min="0" max="65535"/>
   <input id="n008.Max" name="Max" off_x="-100" off_y="100" type="range" min="0" max="65535"/
>
   <output id="n008.Changed" name="Changed" type="event" off_x="100" off_y="-100"/>
```

```
   <output id="n008.Value" name="Value" off_x="100" off_y="-80" type="boolean"/>
   <comment text="Score: " off_x="0" off_y="20"/>
  </component>
  <event id="NewSc" x="30" y="835" mode="input" io_pin="0">
   <comment text="" off_x="0" off_y="20"/>
  </event>
  <arc id="a042" type="read" source="NewSc" target="n008.Set"/>
  <signal id="Score" x="30" y="870" mode="input" type="range" min="0" max="10000" value="0"
dynamic="none" frac="0"/>
  <arc id="a043" type="read" source="Score" target="n008.NewVal"/>
  <operation id="o044" x="30" y="905" constant="1" rot="0" shape="rect" size="20"
locked="true">
   <name off_x="-15" off_y="-5" text="1"/>
   <output off_x="20" off_y="0" name="K" id="o044.K" type="range" min="1" max="1"
dynamic="none" frac="0">
    <expression>
     <text>1</text>
     <operand type="literal" value="1"/>
    </expression>
   </output>
  </operation>
  <arc id="a045" type="read" source="o044.K" target="n008.Visible"/>
  <operation id="o046" x="30" y="975" constant="650" rot="0" shape="rect" size="20"
locked="true">
   <name off_x="-15" off_y="-5" text="650"/>
   <output off_x="20" off_y="0" name="K" id="o046.K" type="range" min="650" max="650"
dynamic="none" frac="0">
    <expression>
     <text>650</text>
     <operand type="literal" value="650"/>
    </expression>
   </output>
  </operation>
  <arc id="a047" type="read" source="o046.K" target="n008.X"/>
  <operation id="o048" x="30" y="1010" constant="640" rot="0" shape="rect" size="20"
locked="true">
   <name off_x="-15" off_y="-5" text="640"/>
   <output off_x="20" off_y="0" name="K" id="o048.K" type="range" min="640" max="640"
dynamic="none" frac="0">
    <expression>
     <text>640</text>
     <operand type="literal" value="640"/>
    </expression>
   </output>
  </operation>
  <arc id="a049" type="read" source="o048.K" target="n008.Y"/>
  <component id="c005" class="ui/checkbox.xml" x="540" y="1000" width="180" height="200"
rot="0" implementation="iopt-flow" target="external" res_location="-" param_string="-">
   <name off_x="-90" off_y="-105" text="ui_checkbox_pause"/>
   <source_model file="files/ui_checkbox.xml"/>
   <input id="c005.Set" name="Set" off_x="-90" off_y="-80" type="event"/>
   <input id="c005.Reset" name="Reset" off_x="-90" off_y="-60" type="event"/>
   <input id="c005.Visible" name="Visible" off_x="-90" off_y="-40" type="boolean"/>
   <input id="c005.Sensitive" name="Sensitive" off_x="-90" off_y="-20" type="boolean"/>
   <input id="c005.X" name="X" off_x="-90" off_y="0" type="range" min="0" max="1023"/>
   <input id="c005.Y" name="Y" off_x="-90" off_y="20" type="range" min="0" max="1023"/>
   <input id="c005.Width" name="Width" off_x="-90" off_y="40" type="range" min="0"
max="1023"/>
   <input id="c005.Height" name="Height" off_x="-90" off_y="60" type="range" min="0"
max="1023"/>
   <input id="c005.PageNr" name="PageNr" off_x="-90" off_y="80" type="range" min="0"
max="31"/>
   <output id="c005.Changed" name="Changed" type="event" off_x="90" off_y="-80"/>
   <output id="c005.Checked" name="Checked" off_x="90" off_y="-60" type="boolean"/>
   <comment text="Pause" off_x="0" off_y="20"/>
  </component>
  <operation id="o050" x="400" y="935" constant="1" rot="0" shape="rect" size="20"
locked="true">
   <name off_x="-15" off_y="-5" text="1"/>
   <output off_x="20" off_y="0" name="K" id="o050.K" type="range" min="1" max="1"
dynamic="none" frac="0">
    <expression>
     <text>1</text>
     <operand type="literal" value="1"/>
    </expression>
   </output>
  </operation>
  <arc id="a051" type="read" source="o050.K" target="c005.Visible"/>
```

```
 <operation id="o052" x="400" y="970" constant="1" rot="0" shape="rect" size="20"
locked="true">
   <name off_x="-15" off_y="-5" text="1"/>
   <output off_x="20" off_y="0" name="K" id="o052.K" type="range" min="1" max="1"
dynamic="none" frac="0">
    <expression>
     <text>1</text>
     <operand type="literal" value="1"/>
    </expression>
   </output>
 </operation>
 <arc id="a053" type="read" source="o052.K" target="c005.Sensitive"/>
 <operation id="o054" x="400" y="1005" constant="50" rot="0" shape="rect" size="20"
locked="true">
   <name off_x="-15" off_y="-5" text="50"/>
   <output off_x="20" off_y="0" name="K" id="o054.K" type="range" min="50" max="50"
dynamic="none" frac="0">
    <expression>
     <text>50</text>
     <operand type="literal" value="50"/>
    </expression>
   </output>
 </operation>
 <arc id="a055" type="read" source="o054.K" target="c005.X"/>
 <operation id="o056" x="400" y="1040" constant="640" rot="0" shape="rect" size="20"
locked="true">
   <name off_x="-15" off_y="-5" text="640"/>
   <output off_x="20" off_y="0" name="K" id="o056.K" type="range" min="640" max="640"
dynamic="none" frac="0">
    <expression>
     <text>640</text>
     <operand type="literal" value="640"/>
    </expression>
   </output>
 </operation>
 <arc id="a057" type="read" source="o056.K" target="c005.Y"/>
 <signal id="Pause" x="655" y="940" mode="output" type="boolean" min="0" max="1" value="0"
dynamic="none" frac="0"/>
 <arc id="a058" type="read" source="c005.Checked" target="Pause"/>
 <component id="s009" class="ui/sound_sample.xml" x="910" y="1000" width="100" height="40"
rot="0" implementation="iopt-flow" target="external" res_location="./files/gui/ui/
pong_if_auto/crash.wav" param_string="-">
   <name off_x="-50" off_y="-25" text="sound_sample_crash"/>
   <source_model file="files/sound_sample.xml"/>
   <input id="s009.Play" name="Play" off_x="-50" off_y="0" type="event"/>
   <output id="s009.Playing" name="Playing" off_x="50" off_y="0" type="event"/>
   <comment text="" off_x="0" off_y="20"/>
 </component>
 <event id="Crash" x="770" y="1000" mode="input" io_pin="0">
   <comment text="" off_x="0" off_y="20"/>
 </event>
 <arc id="a059" type="read" source="Crash" target="s009.Play"/>
 <component id="s010" class="ui/sound_sample.xml" x="170" y="1400" width="100" height="40"
rot="0" implementation="iopt-flow" target="external" res_location="./files/gui/ui/
pong_if_auto/boing.wav" param_string="-">
   <name off_x="-50" off_y="-25" text="sound_sample_boing"/>
   <source_model file="files/sound_sample.xml"/>
   <input id="s010.Play" name="Play" off_x="-50" off_y="0" type="event"/>
   <output id="s010.Playing" name="Playing" off_x="50" off_y="0" type="event"/>
   <comment text="" off_x="0" off_y="20"/>
 </component>
 <event id="Boing" x="30" y="1400" mode="input" io_pin="0">
   <comment text="" off_x="0" off_y="20"/>
 </event>
 <arc id="a060" type="read" source="Boing" target="s010.Play"/>
</net>
```

B. Componente IOPT-Flow da interface do jogo pong, gerado automaticamente a partir da edição e conversão da interface no ambiente IOPT-Flow GUI Editor.

```
<?xml version="1.0"?>
<?xml-stylesheet href="http://gres.uninova.pt/iopt-flow/show-pf.xsl" type="text/xsl"?><net
name="lib">
 <component id="c1" class="local/pong_if_auto.xml" x="150" y="170" width="200" height="240"
rot="0" implementation="iopt-flow" target="default">
  <name off_x="-100" off_y="-125" text="pong_if_auto.html_?"/>
  <source_model file="files/gui/pong_if_auto.xml"/>
  <input id="c1.NewSc" name="NewSc" type="event" off_x="-100" off_y="-100"/>
  <input id="c1.Crash" name="Crash" type="event" off_x="-100" off_y="-80"/>
  <input id="c1.Boing" name="Boing" type="event" off_x="-100" off_y="-60"/>
  <input id="c1.ShowStartBtn" name="ShowStartBtn" off_x="-100" off_y="-40" type="boolean"/>
  <input id="c1.RunGame" name="RunGame" off_x="-100" off_y="-20" type="boolean"/>
  <input id="c1.RunGame2" name="RunGame2" off_x="-100" off_y="0" type="boolean"/>
  <input id="c1.X" name="X" off_x="-100" off_y="20" type="range" min="-1" max="801"/>
  <input id="c1.BaseX" name="BaseX" off_x="-100" off_y="40" type="range" min="-1" max="800"/>
  <input id="c1.Y" name="Y" off_x="-100" off_y="60" type="range" min="-1" max="801"/>
  <input id="c1.ShowGameOver" name="ShowGameOver" off_x="-100" off_y="80" type="boolean"/>
  <input id="c1.Score" name="Score" off_x="-100" off_y="100" type="range" min="0" max="10000"/>
  <output id="c1.StartBtn" name="StartBtn" type="event" off_x="100" off_y="-100"/>
  <output id="c1.ClrGameOver" name="ClrGameOver" type="event" off_x="100" off_y="-80"/>
  <output id="c1.LeftBtn" name="LeftBtn" off_x="100" off_y="-60" type="boolean"/>
  <output id="c1.RightBtn" name="RightBtn" off_x="100" off_y="-40" type="boolean"/>
  <output id="c1.Pause" name="Pause" off_x="100" off_y="-20" type="boolean"/>
 </component>
</net>
```